

SBGAMES 2008

Vii Symposium on Computer Games and Digital Entertainment

*Belo Horizonte, MG - Brazil
november 10th - 12th*



Track:

***Computing
Posters***

www.sbc.org.br/sbgames





november 10th - 12th
Belo Horizonte, MG - Brazil



VII Brazilian Symposium on Computer
Games and Digital Entertainment
November, 10-12, 2008
Belo Horizonte - MG - BRAZIL

PROCEEDINGS
Computing Track - Posters

Published by
Sociedade Brasileira de Computação - SBC

Edited by
Cesar Pozzer
Wallace Lages
Zenilton Patrocínio

Computing Track - Posters Chairs
Cesar Pozzer
Wallace Lages

SBGames 2008 General Chairs
Luiz Chaimowicz
Rosilane Mota

ISBN 857669216-3



9 788576 692164

Universidade Federal de Minas Gerais - UFMG
Pontifícia Universidade Católica de Minas Gerais
PUC Minas

Sponsored by
SBC - Sociedade Brasileira de Computação

Organization



Universidade
Federal de
Minas Gerais



Promotion



Sponsorship

Microsoft

FAPEMIG



globo.com

Google



Table of Contents

SBGames 2008

Preface.....	v
Program Committee	vi
Reviewers	vii

COMPUTING TRACK - Technical Posters

Category: Short Papers

Uso do framework motivacional de Breazeal em NPCs

Marcos Côrtes, Rodrigo Richa, Esteban Clua, Anselmo Montenegro	1-4
---	-----

De Volta para o Futuro: Uma Técnica para Sincronizar Vistas de Jogos Multi-Jogador para Celular via Bluetooth

Lauro Kozovits, Alexandre Sztajnberg, Esteban Clua	5-8
--	-----

Dynamic Game Object Component System For Mutable Behavior Characters

José Ricardo S. Junior, Erick Passos, Esteban Clua, Bruno C. Moreira, Pedro C. Mourão	9-12
---	------

Move-In: Uma API para Interação com Webcam

Jeferson José de Miranda, Marcelo da Silva Hounsell, Alexandre Gonçalves Silva	13-16
--	-------

Algoritmo para Verificação da Solução de Quebra-cabeças Geométricos

Manoel Siqueira Junior, Rafael Alves, Wagner dos Santos, Carol Carvalho, Clayton Reis da Silva, Anselmo Montenegro, Erick Passos, Esteban Clua	17-20
---	-------

Using Navigation Meshes to Improve Storytelling DramatizationVinicius da Costa de Azevedo,
Eduardo C. Dalla Favera,

Cesar Tadeu Pozzer 21-24

Construindo Cenários de Jogos Móveis Utilizando o 3DS Max e a API M3G

Bruno M. Arôxa,

Artur F. Mittelbach 25-28

Estudo e Implementação de Heurísticas para Otimizar os Algoritmos Aplicados a Puzzle Games

Alex Machado,

Esteban Clua,

Carla Bonin,

Gustavo Novaes,

Mauro L. R. A. Filho 29-32

Estratégias de Interação para Storytelling na Televisão Digital

Marcelo Camanho,

Angelo Ciarlini,

Antônio Furtado,

Cesar T. Pozzer,

Bruno Feijó 33-36

Procedural Terrain Generation at GPU Level with Marching Cubes

Bruno José Dembogurski,

Esteban Clua,

Marcelo Bernardes Vieira,

Fabiana Rodrigues Leta 37-40

Arquitetura de Servidor de Xadrez sobre XMPP com Cliente Web

Rubens Suguimoto,

Gabriel Ramos,

Raphael Ribas,

Fabiano Kuss,

Ulysses Bonfim,

Pedro Rocha,

Eduardo Ribas,

Danilo Yorinori

Luis Bona

Fabiano Silva

Alexandre Direne 41-44

RPG Educativo 3D Utilizando Linguagem de Script e Agentes

Rudimar L.S. Dazzi,

Benjamim G. Moreira,

Edmar Souza,

Heitor Adão Jr 45-48

Desenvolvimento de Jogos para o Aperfeiçoamento na Aprendizagem de Disciplinas de Ciência da Computação

Luciano A. Digiampietri,
Diogo D. Kropiwiec 49-52

Simulação de TV Holográfica Utilizando o Wiimote

André Luiz J. D. Tschaffon,
Breno M. de Paula,
Rafael C. Slobodian,
Esteban Clua,
Anselmo Montenegro 53-56

Algoritmo dos Retângulos: um pathfinding para jogos cliente-servidor

Eduardo D. Lima,
Christiano L. Santos 57-60

GAME ENVIRONMENT: Um novo conceito em Design de Interiores

Janiel Almeida,
Virgílio Maia 61-63

Uma abordagem usando algoritmos genéticos em Agentes Inteligentes para Jogos

Felipe José Rocha Vieira,
Christiano Lima Santos 64-67

Pandorga: Uma plataforma open source para a criação e desenvolvimento de jogos

Fábio M. Miranda,
Paulo R. Lafetá,
Leonardo Queiroz,
Carlúcio S. Cordeiro,
Luiz Chaimowicz 68-71

A Framework for Genetic Algorithms in Games

Vinícius Godoy de Mendonça,
Cesar Tadeu Pozzer,
Roberto Tadeu Raittz 72-75

RPG-XEditor: um editor de jogos no estilo RPG baseado em XML

Rodrigo O. da Silva,
Fábio A. C. Bossa,
Rômulo C. Silva,
Felipe H. Moreno,
Eliane N. Pereira,
Izaura M. Carelli,
João A. Fabro 76-79

Criando níveis 3D para jogos móveis usando um editor 2D

Bruno M. Arôxa,
Luiz A. B. da Silva,
Artur F. Mittelbach 80-83

Uma Comparação Preliminar entre Tecnologias para Implementação de Variabilidades em Jogos para Celulares

Rogério Celestino dos Santos,
Marco Túlio Valente 84-87

PREFACE

Welcome to the VII Edition of the Brazilian Symposium on Computer Games and Digital Entertainment, the SBGames 2008. SBGames is the yearly symposium of the Special Interest Group on Games and Digital Entertainment of the Brazilian Computer Society (SBC). SBGames 2008 is the most important event on game research and development to take place in Latin America, promoted by the Brazilian Computer Society (SBC) with the support of the Brazilian Electronic Game Development Companies Association (ABRAGAMES). This year the symposium brings together students, professors, artists, designers and professionals from several universities, research centers, graphical design centers and game industry.

This volume contains the 22 technical posters accepted for the computing posters track. This year we had 27 submissions, 12 of which were originally submitted to the computing track. The evaluation process was double blind and each paper was reviewed by at least 3 experts. All papers with consistently good reviews were accepted. We hope this will help the authors to improve their work with the feedback received and also stimulate young researchers in the field.

The SBGames 2008 is composed by 4 main tracks: Computing, Art & Design, Industry and Games & Culture, 2 festivals (Independent Games and Art Exhibition), Poster Exhibitions, Tutorials, Keynote presentations, and other satellite events. The papers from the other tracks, including the Computing Track, have been included within the Conference Proceedings CD-ROM.

We would like to thank all authors, whose work and dedication made possible to put together an exciting program. Next, we would like to thank all members of the technical program committee and reviewers, for their time helping us maintain the overall quality of the program.

We would like to wish all attendees an exciting symposium!

Belo Horizonte, November 2008,

Cezar Pozzer & Wallace Lages
Chairs of the Program Committee – Computing Posters track

Program Committee

Alessandro Silva	UFMG - Universidade Federal de Minas Gerais
André Brandão	UFF - Universidade Federal Fluminense
Andre Silva	Unicamp
Börje Karlsson	PUC-Rio
Cesar Pozzer	Universidade Federal de Santa Maria
Erick Passos	UFF - Universidade Federal Fluminense
Esteban Clua	UFF - Universidade Federal Fluminense
Fabio Policarpo	Double Fine Productions
Fábio Binder	PUC-Rio
Fernando Bevilacqua	Universidade Federal de Santa Maria
Fernando Marson	Univ. Regional Integrada - Santo Ângelo
Fernando Osório	USP - Universidade de São Paulo
Heitor Costa	Universidade Federal de Lavras
Jacques Brancher	URI - Campus de Erechim
Joao Luiz Campos	PUC-Rio
Manuel M. Oliveira Neto	UFRGS - Univ. Federal do Rio Grande do Sul
Marcelo Dreux	PUC-Rio
Marcelo Malheiros	UNIVATES
Marcio Zuchini	Policamp
Nizam Omar	Universidade Presbiteriana Mackenzie
Rafael Rieder	PUC-RS
Rafael Torchelsen	UFRGS - Univ. Federal do Rio Grande do Sul
Romero Tori	Centro Univ. Senac / Univ. de Sao Paulo
Sérgio Pellegrino	ITA
Silvano Malfatti	Lab. Nacional de Computação Científica
Soraia Musse	PUC-RS
Vinícius Mendonça	Siemens Enterprise Communications Ltda
Wallace Lages	UFMG - Universidade Federal de Minas Gerais
Zenilton Patrocínio Jr	Pontifícia Universidade Católica de Minas Gerais

Reviewers

Adelailson Peixoto
Alessandro Silva
André Campos
André Luiz Brandão
Andre Tavares da Silva
Borje Karlsson
Bruno Feijó
Cesar Pozzer
Claurirton Siebra
Drew Davidson
Edmond Prakash
Erick Passos
Esteban Walter Gonzalez Clua
Fábio Binder
Fabio Policarpo de Oliveira
Fernando Bevilacqua
Fernando Pinho Marson
Fernando Santos Osório
Fernando Trinta
Geber Ramalho
Heitor Augustus Xavier Costa
Henrique Vicentini
Jacques Brancher
Jezer Oliveira
Jim TerKeurst
Joao Luiz Elias Campos
José Saito
Judith Kelner
Kalinka Castelo Branco
Leandro Fernandes
Leandro Motta Barros

Luiz Gonzaga da Silveira Jr
Manuel Menezes Oliveira Neto
Marcelo de Gemensoro Malheiros
Marcelo Dreux
Marcio Henrique Zuchini
Marcio Pinho
Marcio Sarroglia Pinho
Marcos Kich
Maria Andréia Rodrigues
Mario Massakuni Kubo
Michael Youngblood
Nizam Omar
Patrícia Tedesco
Rafael Piccin Torchelsen
Rafael Rieder
Roberto Cezar Bianchini
Rodrigo Hahn
Romero Tori
Samir Souza,
Sérgio R. Pellegrino
Silvano Malfatti
Sílvio Cazella
Silvio Melo
Solon Rabello
Soraia Musse
Vinícius Godoy de Mendonça
Vitor Pamplona
Waldemar Celes
Wallace Santos Lages
Zenilton K G Patrocínio Júnior

SBGAMES 2008

COMPUTING TRACK

TECHNICAL POSTERS

Uso do *framework* motivacional de Breazeal em NPCs

Marcos Côrtes

Rodrigo Richa

Esteban Clua

Anselmo Montenegro

marcos.v.p.cortes@gmail.com rodrigoricha@gmail.comesteban@ic.uff.branselmo@ic.uff.br

Universidade Federal Fluminense – Instituto de Computação, Brasil

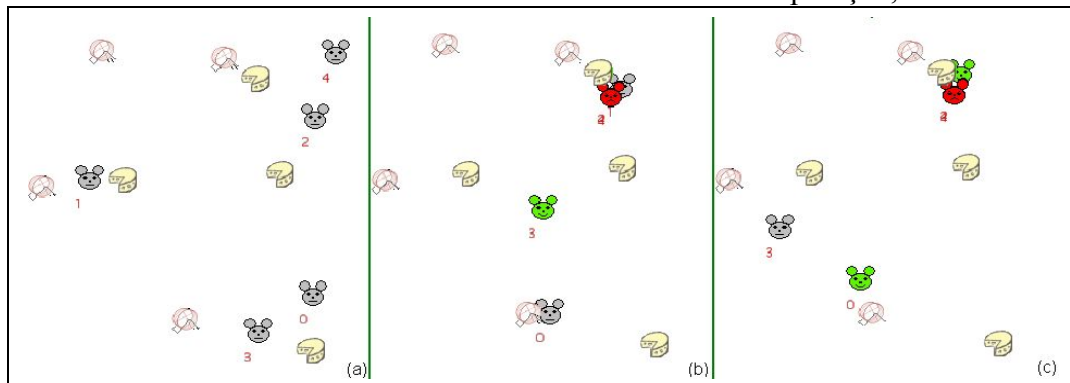


Figura 1: Imagem do protótipo de simulador de NPCs usando o *framework* de Breazeal exibindo instantes (a) 6 segundos, (b) 19 segundos e (c) 23 segundos da mesma simulação do gráfico 1

Resumo

Atualmente, tecnologias de entretenimento como games e TV digital possuem maior complexidade e exigem maior realismo dos seus personagens (maior “ilusão de vida”). Métodos tradicionais em IA de jogos para controle de NPCs, como máquina de estados, embora sejam suficientes para algumas tarefas de vida artificial, são pouco escaláveis e flexíveis, pois precisam de um modelo rígido e lógico do objeto simulado. Este artigo descreve o uso de um sistema motivacional baseado no *framework* de Breazeal na inferência de movimento e comunicação de NPCs, sendo uma abordagem alternativa e mais flexível no escopo dos jogos digitais.

Palavras-chave: Atores sintéticos, Inteligência Artificial, Vida Artificial, Modelo de Emoção, NPC, Agentes.

1. Introdução

A simulação de NPCs em jogos digitais são normalmente implementados através de máquina de estados e agentes. [POZZER 03] argumenta que com os sistemas de simulação interativos exigem cada vez mais sistemas de IA mais escaláveis e complexos. [SILVA 02] também sugere a necessidade de buscar alternativas aos métodos clássicos de controle de NPCs.

Com isto em mente, propomos neste trabalho um agente/NPC controlado sistema motivacional com emoções que influem no seu comportamento e gerar a comunicação entre eles. Usamos como base o

framework de [BREAZEAL 98].

O protótipo foi construído para ser genérico, permitindo a um game designer ou especialista de domínio possa especificar os requisitos deste agente e possa implementá-los no motor motivacional.

2. Framework de Breazeal

O sistema de [BREAZEAL 98] consiste em um robô chamado *Kimet* que simula a interação de uma criança com seu tutor (*infant-caretaker dyad*). O objetivo do agente robô é aprender uma forma eficiente de obter suas demandas junto ao tutor humano através de suas expressões faciais, que são representações externas de suas emoções.

O *framework* é composto por três classes de entidades: *drives* (pulsões), *emoções* e *comportamentos*. Todas estas entidades são consideradas processos (*process*) implementados como um operador (um *transducer*) como na equação 1. O processo recebe n entradas i , sendo que cada uma possui um peso w_i , podendo ser este positivo ou negativo. O somatório destes valores somado a uma interferência b (*bias*) gera o valor de saída. Este valor de saída é chamado de **carga**, **energia** ou simplesmente **valor**. As entradas i de um processo são normalmente valores de carga de outro processo. A ligação de um processo a outro é chamada de **influência**.

Cada componente está sujeito a “ser ativado” quando extrapola um limiar (*threshold*) superior (ou inferior). Quando isto ocorre, o processo ativa uma computação para tal evento, como por exemplo influenciar determinado comportamento [MAES 90].

$$x = \sum_{j=0}^n w_j \cdot i_j + b$$

Equação 1: Operador (*transducer*)

3. Implementação proposta

A implementação proposta considerou que todas as três entidades originais derivam de um processo. Este é discretizado de tal forma que a cada tempo t , seja executada a Equação 1 considerando os valores de i do tempo anterior ($t-1$).

O processo deve possuir um limiar superior e um inferior. Quando um desses eventos ocorre, o processo passa a influenciar outro. Isto é: Se o processo “fome” extrapolar o limiar superior, ele passa a influenciar outro processo, o “comer”.

Além disso, adicionamos mais uma classe de itens: “Sensor”. Este permite a ligação do sistema com o mundo externo. Basicamente eles são versões simplificadas de um processo que tem seus valores dependentes do estado do ambiente.

No final, teremos uma “rede” (grafo) de processos que se influenciam partindo dos sensores e indo até os comportamentos, que gerarão o controle necessário ao NPC.

A Tabela 1 demonstra mais detalhadamente quais os processos definidos e as relações adotadas. Ela foi constituída pensando nos requisitos de simular nosso problema abordado no protótipo (uma comunidade de camundongos).

Devemos o *framework* é **simbólico** e modela o funcionamento fisiológico de seres vivos. Não devemos confundí-lo como um modelo conexionista (ex. rede neural). A teoria adotada por ambos os modelos são diferentes e incompatíveis.

3.1 Drive

Um *drive* como um impulso natural (essencial) **que deve sempre** ser satisfeito pelo agente. Sua função é simular necessidades fisiológicas naturais de um ser vivo que aumentam com o decorrer do tempo devido ao consumo do organismo de seus recursos (exemplo: fome e sede).

Quando o valor de um *drive* atinge um limiar, ele entra para o estado ativo e precisa ser satisfeito. Isto equivale influenciar um comportamento que o satisfará.

Um exemplo de *drive* seria a fadiga: A locomoção do agente e gastos cognitivos incrementariam este *drive*. Quando ele excede um limiar é ativado, influenciando o “desejo de dormir” do agente para satisfazer o *drive* ativado.

Para os *drives*, os limiares de ativação podem ser inferiores e/ou superiores.

3.2 Comportamento

Comportamentos são como atividades (ações, reações, padrões) observáveis num ser vivo e também considerados como uma mensagem do agente para todos os elementos do ambiente. Um comportamento que satisfaz um *drive* é um comportamento consumidor (*consummatory behavior*) deste *drive*.

Existe uma outra classe de comportamento: os facilitadores (*appetitive behaviors*). Cada comportamento consumidor possui um ou mais comportamentos facilitadores. Quando um comportamento consumidor não pode ser realizado por algum motivo, tal como a ausência de variáveis para ele se realizar ou algum impedimento, o sistema de inferência tem em mãos os comportamentos facilitadores para mudar o estado do ambiente e do agente para um outro estado apto a execução do comportamento consumidor.

Um comportamento varia seu valor de zero até infinito e possui somente o limiar superior. Quando o seu valor extrapola o limiar superior, ele é ativado e o comportamento passa a ser efetuado, se possível. E um comportamento pode ser executado junto com outro, pelo simples fato de que foram especificados de tal forma que um não anula a presença do outro (não existe um comportamento andar e outro correr), salvo o comportamento de dormir, que inibe todos os outros.

Por último, os comportamentos são implementados associados a um algoritmo de “tentar executar” (*try play*) que sempre é chamada quando ele extrapola o limiar superior. Esta função realiza a ação do agente no ambiente e determina uma especialização do comportamento. Caso ele não consiga executar por alguma razão (como por exemplo, comportamento de comer ser inviável, pois não há comida perto o suficiente), o comportamento tenta executar os seus comportamentos facilitadores.

3.3 Emoção

A emoção também é um processo. Entretanto em cada passo de cálculo é decrementado de um valor definido. Sempre que for extrapolado o seu limiar superior ela é ativada. Caso mais de uma emoção seja ativada, apenas será exibida a que possuir maior valor. Ela é exibida através da mudança do desenho do NPC.

A implementação proposta utiliza um modelo baseado em Elkan e Plutchik, tal como o framework de [BREAZEAL 98] (na verdade usamos um subconjunto de itens do modelo original de [BREAZEAL 98]). Estes modelos são classificados como modelos de “emoções puras” ou “emoções discretas”. Estes modelos determinam que as emoções sejam compostas por um número limitado de emoções primitivas discretas e disjuntas.

3.4 Sensor

Para que o agente possa ter percepção do mundo e gerar a comunicação entre os agentes, foi adicionado ao sistema o conceito de sensores. Estes sensores podem servir de entrada aos processos e possuem duas funções básicas: podem inferir algo em relação ao mundo (função ativa) e receber e armazenar alguma mensagem enviada ao agente (função passiva). O primeiro caso corresponde ao sensor que busca itens no ambiente, como comida ou outro agente. O segundo são receptores que podem receber mensagens de outros agentes. Assim, qualquer alteração provocada de um agente para o outro deve ser propagada por uma mensagem do agente causador ao agente afetado.

No segundo caso, estes sensores podem servir de influência a processos. O valor desta influência é dado como na equação 1, porém a incógnita i é a intensidade de uma mensagem j . Em outras palavras, a influência de um sensor passivo é dada pelo somatório das intensidades das mensagens capturadas.

Com isto, foi possível gerar interações entre os NPCs, onde eles podiam conversar e brigar um com os outros.

4. Calibração e Resultados

Depois de implementado, o sistema precisou ser calibrado. Isto consiste em mudar os valores de pesos do sistema de tal forma que apresentasse um comportamento satisfatório, entenda-se como satisfatório o fato de transmitir ilusão de vida, tema já discutido no início do artigo.

Para calibrações ruins o problema mais comum que ocorreu é uma tendência aos agentes a se comportarem de uma mesma forma, convergindo para um único tipo de emoção, como por exemplo, todos ficarem com raiva. Ou ainda apenas alguns *drives* ou emoções serem ativados em detrimento de outras.

Com algumas tentativas foi possível chegar em uma situação balanceada, onde todos os processos puderam ser ativados e os agentes apresentaram situações de correlação entre suas ações, tais como:

1. Um agente A fica com raiva por que não achou um brinquedo e começa a agredir o agente B. O agente B por sua vez fica triste ou com medo;
2. Um grupo de agentes fica feliz, pois estão mais próximos de um brinquedo e uma fonte de comida;
3. Agentes alegres ficam juntos, pois estão realizando o comportamento de conversar.

4.1 Resultados

Para fins de análise, se acompanhou o sistema motivacional de um NPC (o agente 0), conforme o

gráfico 1.

Desde o início do tempo, percebe-se o *drive* “vontade de brincar” em crescimento. Chegando no instante 6 (figura 1a) ele é ativado e passa a influenciar o comportamento brincar. Este em algum momento é ativado e força o agente a ir atrás de um brinquedo para satisfazer o *drive*. No instante 19 (figura 1b) isto ocorre e o *drive* “vontade de brincar” é satisfeito: como está negativo, extrapola seu limiar inferior e passa a influenciar a emoção “Feliz”. Esta é incrementada e no pico da curva, no instante 25 (figura 1c), o agente esta exibindo uma caricatura feliz.

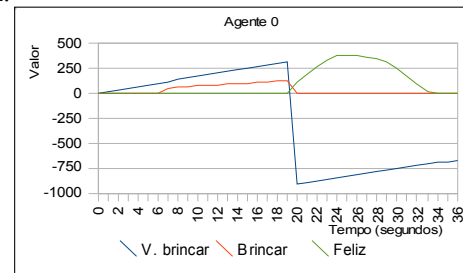


Gráfico 1: Gráfico de alguns processos em do agente 0 em determinada simulação

5. Considerações finais

O presente trabalho demonstra uma possível implementação para simulação de NPCs com inferência sobre suas necessidades através da adaptação do *framework* de Breazeal para este problema. Este método é genérico o suficiente para que se possa aplicá-lo a várias naturezas de agentes, desde animais, NPCs de RPG ou outros.

Verificou-se que a técnica proposta por [BREAZEAL 98] permite uma boa solução para simulação de NPCs. O grande ganho deste trabalho é usar o modelo na geração de movimento dos NPCs e sua interação (comunicação). Obtendo-se razoável “ilusão de vida” com baixo custo computacional e um algoritmo relativamente simples.

Referências:

- [BREAZEAL 98] Breazeal, C., 1998. A motivational system for regulating human-robot interaction. Proceedings of the fifteenth National Conference on Artificial Intelligence, Madison, EUA, pp54-61.
- [BULLOWA 79] Bullowa, M., 1979. Before Speech: The Beginning of Interpersonal Communication, Cambridge University Press, Cambridge, London [in RODRIGUES].
- [CAÑAMERO 97] Cañamero, L., 1997. A Hormonal Model of Emotions for Behavior Control. Proceedings of the 4th European Conference on Artificial Life (ECAL'97). [in MORGADO 05].
- [DAMASIO 94] Damásio, A., 1994. O Erro de Descartes. Europa-América [in ZAGALO 04].

- [FURTADO 02] Furtado, A. L.; Ciarlini, A. E. M. 2002. Cognitive and affective motivation in conceptual modelling. Avaliado em: http://editorial.unab.edu.co/revistas/rcc/pdfs/r32_art1_c.pdf [em 01 de maio de 2008].
- [GUDWIN 08] Gudwin, R.. Agentes com Emoções. Avaliado em: <http://www.dca.fee.unicamp.br/~gudwin/courses/IA009/> em 20/05/2008.
- [MAES 90] Maes, P., 1990. Learning Behavior Networks from Experience, ECAL90 [in BREAZEL].
- [MORGADO 05] Morgado, Luís Filipe Graça, 205. Integração de Emoção e Raciocínio em Agentes Inteligentes. Universidade de Lisboa, Lisboa, Portugal.
- [POZZER 03] Pozzer, C. Furtado, A.F.; Ciarlini, A. E. M., 2003. Agentes e emoções em histórias interativas. Departamento de Informática da PUC-RJ, Rio de Janeiro, Brasil. Avaliado em: ftp://ftp.inf.puc-rio.br/pub/docs/techreports/03_39_pozzer.pdf [em 20 de abril de 2008].
- [ORTONY 88 *at al*] Ortony, A.; Clore, G.; Collins, A., 1988. The Cognitive Structure of Emotions Cambridge University Press. [in MORGADO].
- [SEP 07] SEP, 2007. Emotion. Artigo da Stanford Encyclopedia of Philosophy. Acessado em: <http://plato.stanford.edu/entries/emotion> [em 05 de maio de 2008].
- [SILVA 02] Silva, D. R. D.; Tedesco, P. R. T.; Ramalho G. L., 2006. Usando Atores Sintéticos em Jogos Sérios: O Case SmartSim. SBGames 2006, Pernambuco, Brasil. Avaliado em: <http://cin.ufpe.br/~sbgames/proceedings/files/Usando%20Atores.pdf> [em 01 de maio 2008].
- [RODRIGUES 07], Rodrigues P. S. L., 2007. Um sistema de geração de expressões faciais dinâmicas em animações faciais 3D com processamento de fala. Tese de doutorado, Departamento de Informática da PUC-RJ, Rio de Janeiro, Brasil.
- [VELÁSQUEZ 98] Velásquez, J., 1998. Modeling Emotion-Based Decision-Making. Proceedings of the 1998 AAAI Fall Symposium Emotional and Intelligent: The Tangled Knot of Cognition, 164-169. AAAI Press.
- [ZAGALO 04] Zagalo, N., Branco, V., Barker, A., 2004, Emoção e Suspense no Storytelling Interativo, in Actas, Games2004 - Workshop Entretenimento Digital e Jogos Interactivos, Lisboa, Portugal, (pp.75-84)

Influência / Influenciado	Medo	Raiva	Feliz	Triste	Fome	V. de brincar	V. de conversar	Dor de barriga	Cansaço	Comer	Brincar	Dormir	Conversar	Bater	Defecar	Move Random
Medo	0	-,O	0	0	0	0	0	0	0	0	0	0	0	-,O	0	0
Raiva	0	0	0	-,O	0	0	0	0	0	0	0	0	0	+,O	0	0
Feliz	0	0	0	0	0	0	0	0	0	0	0	0	0	-,O	0	0
Triste	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fome	+,O	0	0	0	0	0	0	0	0	+,O	0	0	0	0	0	0
V. de brincar	0	0	0	-,U	0	0	0	0	0	0	+,O	0	0	0	0	0
V. de conversar	0	-,O	-,U	0	0	0	0	0	0	0	0	0	+,O	-,O	0	0
Dor de barriga	0	+,O	-,U	+,O	0	0	0	0	0	0	0	0	0	0	+,O	0
Cansaço	0	0	0	0	0	0	0	0	0	0	0	+,O	0	0	0	0
Comer	0	+,O	0	0	0	0	0	0	+,O	0	0	0	0	0	0	-,E
Brincar	0	0	0	0	0	0	0	0	+,O	0	0	0	0	0	0	-,E
Dormir	0	0	+,U	0	0	0	0	0	0	0	0	0	0	0	0	-,E
Conversar	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-,E
Bater	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-,E
Defecar	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Move Random	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Legenda																
0	Não há relação															
+	A influência é positiva															
-	A influência é negativa															
U	A influência ocorre quando o influenciador extrapolar um limiar inferior (under_state)															
O	A influência ocorre quando o influenciador extrapolar um limiar superior (over_state)															
E	A influência ocorre em ambos os limiares															

Tabela 1: Tabela do grafo de relação entre processos do sistema motivacional

De Volta para o Futuro: Uma Técnica para Sincronizar Vistas de Jogos Multi-Jogador para Celular via Bluetooth

Lauro E. Kozovits Alexandre Sztajnberg Esteban G. Clua*

Universidade Estadual do Rio de Janeiro, DICC-IME, Brazil
Fluminense, IC-UFF, Brazil

*Universidade Federal

Abstract

The scene view synchronization in different workstations both in simulations and in multiplayer games is a research topic of great relevance in both areas. This work presents an alternative and yet simple synchronization technique for Bluetooth connected mobile multiplayer games.

Keywords: mobile games, multiplayer games, bluetooth games, dead reckoning

Authors' contact:

{lauro, alexszt}@ime.uerj.br
**esteban@ic.uff.br

1. Introdução

Jogos multi-jogador constituem uma sub-área de ambientes virtuais ligados por rede (net-VEs) que têm despertado um grande interesse de pesquisas tanto por parte da indústria quanto pela comunidade científica. Trabalhos relativos a otimização e tratamento conveniente de mensagens trocadas nestes ambientes demonstram a relevância da pesquisa nesta área [Kozovits 2004].

A interatividade e a sincronização da visualização de cenas por parte de cada participante em jogos multi-jogador e simulações é um tópico de pesquisa tanto nas áreas de simulação quanto de jogos. Em um jogo multi-jogador torna-se particularmente importante que um momento de pontuação seja claramente acusável por todos os participantes ou o jogo perderá a credibilidade e o interesse.

O problema crítico de consistência visual num jogo multi-jogador ocorre quando a cena vista por um jogador é diferente daquela vista por outro (ou outros), jogador (jogadores). Por exemplo, é possível que uma bola ultrapasse um goleiro em uma workstation enquanto em outra a bola ainda está chegando de modo que o defensor ainda tenha a chance de capturá-la.

O problema relatado é freqüente em função do fato de que há atrasos no envio de mensagens entre pontos de uma rede seja ela Internet, intranet ou mesmo Bluetooth como tratada no presente trabalho.

Jogos multi-jogador para celular fazem parte de um mercado em franca expansão [Kozovits 2008] e podem

apresentar características especiais, eventualmente demandando soluções um pouco diferentes daquelas usualmente adotadas para ambientes PC, devido à diferença capacidade de processamento de suas CPUs, memória disponível para os programas, etc.

Neste trabalho apresentamos uma técnica simples que possibilita alcançar uma maior consistência visual entre dispositivos executando jogos multi-jogador para celular via Bluetooth. Na Seção 2 apresentamos os trabalhos relacionados. Na Seção 3 a técnica proposta é apresentada. Na Seção 4 seus resultados são avaliados e na Seção 5 a conclusão é descrita.

2. Trabalhos Relacionados

Em ambientes PC e em workstations em geral é comum o uso da técnica de dead-reckoning [IEEE Standard 1278, 1993], [Singhal e Cheriton] e [Singhal e Zyda] para possibilitar a previsão de trajetória de forma mais suave de objetos em movimento evitando-se saltos ou descontinuidades entre uma mensagem de atualização e outra, bem como evitar o envio de um número elevado dessas mensagens por unidade de tempo.

Por exemplo, se um avião voando durante uma simulação fizer um desvio de sua trajetória e uma ou mais mensagens forem perdidas é possível que numa workstation ele apareça num vôo direto e em outra ele seja mostrado com a trajetória contemplando o desvio (figura 1a).

O algoritmo de dead-reckoning calcula uma trajetória suave (T2) até o ponto de convergência correspondente à mensagem P3 na figura. Assim, equações não lineares são geralmente empregadas para ajustar as coordenadas recebidas, resultando num visual mais crível e, portanto, mais compatível com um vôo real.

Entretanto, se por um lado o uso de dead-reckoning implica numa taxa menor de mensagens enviadas numa simulação ou jogo se houver um obstáculo na trajetória T2 há um problema evidente para a simulação, pois uma colisão pode ser mostrada numa workstation, mas não em outra ou mesmo uma situação irreal de colisão sem dano (problema de superposição) pode ocorrer comprometendo a qualidade do jogo ou da simulação.

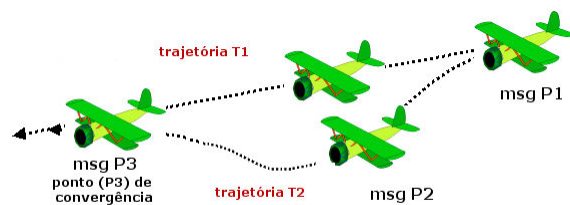


Figura 1a: Não-consistência visual entre duas trajetórias mostradas em duas workstations (T1 e T2) em função da perda de mensagem de posição P2. A chegada da mensagem P3 e o algoritmo de dead-reckoning permitem uma trajetória de compensação suave.

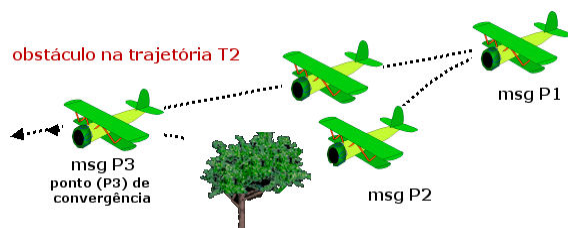


Figura 1b: Se houver um obstáculo na trajetória T2 um problema pode ser criado para a simulação.

O uso de técnicas de dead-reckoning é um padrão em jogos de ação via Internet ou mesmo em simulações militares [UNITED STATES DEPARTMENT OF DEFENCE 2002].

Uma outra abordagem é o uso de agentes inteligentes que tomam decisões baseados em metas a serem alcançadas [Szwarcman et. al. 2000]. Neste trabalho não há, na prática, uma consistência visual a cada fotograma, mas sim de resultados finais.

Por exemplo, de acordo com o que ilustra a figura 2, se um personagem deve sair de uma sala A em primeiro plano e efetivamente chegar a uma sala B ao fundo (esta é a meta) então não faz diferença neste tipo de simulação qual o caminho tomado para se chegar àquela situação final (personagem na sala B). Se houver duas portas (digamos uma à direita e outra à esquerda) que possam conduzir o personagem da sala A até a sala B é possível que o personagem orientado a metas possa, na workstation AZUL (lado esquerdo da figura 2), escolher sair pela porta da esquerda.

Na workstation ROXA (lado direito da figura 2), em função de um atraso na rede, um objeto em movimento (por ex. um outro personagem) pode estar momentaneamente obstruindo a trajetória para a porta da esquerda nesta workstation ROXA (o que não ocorre na workstation AZUL). O tipo de simulação (baseada em agentes reativos orientados a metas) conduzida na workstation ROXA irá, naquele momento, e, em função daquela sua situação peculiar, optar por chegar ao mesmo destino final (sala B) saindo entretanto pela porta da direita.

Portanto, apenas a situação final é visualmente consistente: nas workstations AZUL e ROXA o personagem foi da sala A para a sala B. Situações intermediárias visualmente inconsistentes são irrelevantes para o objetivo da simulação como um todo. É importante observar que isto pode ser aplicável para um dado jogo de estratégia por exemplo, mas nem sempre é aplicável a um jogo de ação ou arcade para celular.

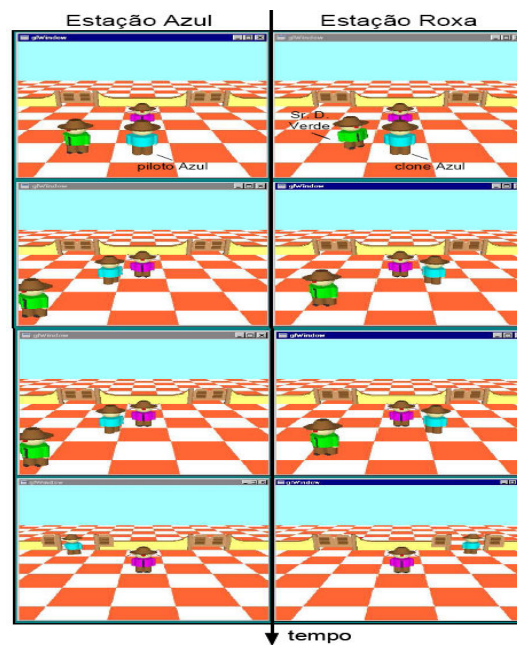


Figura 2: ilustração do mecanismo de personagens reativos orientados por metas extraído de [Szwarcman et. al. 2000]

3. A Técnica Proposta

Considere-se um jogo multi-jogador [Showball 2008], como na Figura 3a, em que dois celulares, um atuando como servidor e outro como cliente utilizam a interface Bluetooth para comunicação. Observa-se que, é necessário um ajuste para compatibilizar o visual de objetos em movimento rápido visto na tela de dois celulares.

Nossa abordagem explora a constância no atraso de uma rede Bluetooth. Além disso, não se observou nenhum problema de *lag* que são comuns em redes não locais como a Internet por exemplo. Assim, torna-se possível estimar a posição de um objeto em movimento tal como será visto pelo outro ponto.

Considere-se, por exemplo, que no início de uma interação o servidor gera a coordenada do objeto em movimento, digamos $(X_{t=initial}, Y_{t=initial})$ e a envia para o cliente. No momento em que o cliente recebe esta coordenada e a exibe, o servidor já estaria mostrando o objeto numa coordenada futura $(X_{t=futuro}, Y_{t=futuro})$ gerando uma discrepância entre os dois visuais.

Isto torna-se visualmente mais relevante quanto maior for a velocidade do objeto. Na Figura 3a isto é ilustrado através de dois celulares, lado a lado, conectados por Bluetooth. Observa-se que a posição da bola no servidor (celular da esquerda) está ligeiramente adiantada em relação ao cliente (celular da direita).

Observe ainda que o jogo Showball foi feito de modo que um jogador possa se posicionar de frente um para o outro. A barrinha inferior em preto representa o jogador local enquanto a barrinha em vermelho representa o jogador remoto.

Na técnica proposta, o cliente retransmite para o servidor a coordenada que ele (cliente) recebeu do próprio servidor (Figura 3b). O servidor irá então recebê-la num tempo $t=atual$ quando o objeto (no servidor) já deveria estar na posição $(X_{t=atual}, Y_{t=atual})$.

Assumindo-se que o atraso na rede bluetooth seja semelhante nos dois sentidos, torna-se possível esperar que a informação de posição vista pelo cliente seja aproximadamente o valor médio entre a posição atual calculada pelo servidor $(X_{t=atual}, Y_{t=atual})$ e a posição informada pelo cliente $(X_{t=inicial}, Y_{t=inicial})$.

No tempo $t=atual$, o servidor irá enviar como nova mensagem de atualização para o cliente a posição atual, mas irá desenhar o objeto numa posição média numa tentativa de compensar os atrasos de transmissão nos dois sentidos e reproduzir o que é desenhado no cliente no mesmo momento:

$$(X_{\text{exibido servidor}}, Y_{\text{exibido servidor}}) = [(X_{t=atual}, Y_{t=atual}) + (X_{t=inicial}, Y_{t=inicial})] / 2.$$

A posição simulada pelo servidor é a posição futura que será desenhada pelo cliente. Entretanto, este mesmo servidor irá, para efeito apenas de visualização, retornar a uma posição anterior. Assim, usamos a expressão “de volta para o futuro” para denominar esta técnica. Na figuras 3a mostramos a posição simulada na cor preta, na figura 3b a posição anterior (a que retorna) em vermelho e, finalmente, em 3c a média das posições (em verde). Esta média corresponde, aproximadamente àquela que é vista pelo cliente (lado direito da figura) gerando uma maior consistência visual.



Figura 3a: jogo Showball executando no simulador e exibindo a bola em movimento (em preto) gerada no celular Bluetooth executando como servidor (celular da esquerda). A

posição com ligeiro atraso de transmissão (em verde) é mostrada no celular cliente Bluetooth à direita.



Figura 3b: jogo ShowBall executando no simulador e exibindo a bola em movimento (em vermelho) no servidor tal como informada pelo cliente.

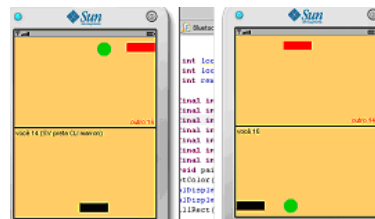


Figura 3c: jogo ShowBall executando no simulador e exibindo a bola em movimento (em verde) no servidor (à esquerda) em posição semelhante aquela que é vista no cliente (à direita) compensando, aproximadamente, os atrasos da rede Bluetooth (de ida e de volta) e gerando uma maior consistência visual.

4. Análise de Resultados

Foi implementado o programa Showball [2008] usando-se Java ME para demonstrar a técnica proposta. A técnica apresentada assume que o atraso entre o que é enviado do servidor ao cliente é semelhante aquele no sentido contrário. Isto pode não ser verdade o que poderia resultar numa discrepância entre a posição vista no cliente e a posição estimada no servidor com a técnica proposta.

Outro fato que cabe relatar é que a técnica proposta efetivamente ajusta o visual entre os dois pontos, mas não elimina acelerações ou retardos na velocidade do da bola que foram observados afetando a fluidez de seu movimento em situações esporádicas. Entretanto, o protocolo JSR 82 [2002] não nos permite ter o controle das camadas mais inferiores deste protocolo para investigar o que estava efetivamente ocorrendo nos testes realizados.

Ainda que possa haver variações entre os atrasos foi observado empiricamente que mesmo o uso da média aritmética entre as duas posições produz resultados melhores nos telefones testados do que o seu não uso. Sendo uma média simples, o benefício, na avaliação dos autores supera o seu não uso.

A técnica formal de dead reckoning [Singhal e Cheriton 1994] poderia, eventualmente, trazer melhores resultados ao custo de uma complexidade bem maior do jogo para celular o que pode ser um fator limitante de seu uso.

Cabe finalmente observar que o problema de inconsistência visual em jogos para celulares via Bluetooth é relevante apenas em objetos com velocidades elevadas.

O jogo Showball [2008] permite que se acelere a bola rebatida de modo a evidenciar o problema do atraso tratado deste trabalho. Entretanto, observou-se que para velocidades compatíveis com a resposta humana mantendo-se a jogabilidade do produto as diferenças de posição perceptíveis geradas devido ao atraso são pequenas.

Na figura 4 mostra-se a execução do jogo Showball implementado, evidenciando-se que quanto mais rápido (par superior na figura) é o movimento de um objeto, mais discrepante visualmente é o seu posicionamento entre os celulares participantes do jogo.

No par inferior quando a velocidade é reduzida a coordenada da bola gerada originalmente no servidor (em preto) não apresenta uma diferença tão expressiva em relação a coordenada informada (de volta, em vermelho) pelo cliente e, por último, àquela calculada (em verde) com a técnica proposta.

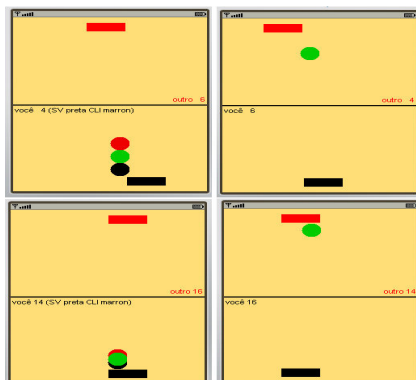


Figura 4: Showball em diferentes velocidades de execução

5. Conclusão

A técnica proposta “de volta para o futuro” é simples, de fácil implementação e eficaz o que é relevante na programação de jogos para celular. Funciona bem para jogos multi-jogador utilizando Bluetooth.

Possivelmente, a implementação de uma técnica formal de dead-reckoning poderia gerar resultados mais acurados. Entretanto, o esforço nesta implementação, além de aumentar o tamanho do código, algo crítico para um jogo para celular, ainda poderia representar uma precisão irrelevante para jogos multi-jogador via Bluetooth.

Trabalhos futuros devem estudar a escalabilidade da técnica proposta em jogos com um número maior de jogadores simultâneos.

A implementação do jogo Showball utilizando a tecnologia proposta está disponível para download e estudo em <http://www.jogos.etc.br/showball>

Agradecimentos

O autor gostaria de agradecer o programa de professores visitantes da UERJ e o apoio dado pelo Instituto de Matemática e Estatística da mesma Instituição IME-UERJ.

Referências

- BLUETOOTH 2008. *Bluetooth Core Specification v2.1* [online]. Disponível em: <http://www.bluetooth.com/Bluetooth/Technology/> [Acessado em 15 de Agosto de 2008].
- IEEE STANDARD 1278 - 1993. IEEE standard for information technology - Protocols for distributed simulation applications: entity information and interaction, IEEE Computer Society
- JSR 82, 2002. Java APIs for Bluetooth. [online]. Disponível em: <http://jcp.org/en/jsr/detail?id=82> . [Acessado em 15 de Agosto de 2008].
- KOZOVITS, L.E., 2004. *Otimização de Mensagens e Balanceamento de Jogos Multi-Jogador*. Tese de doutorado, Instituto de Informática, PUC-Rio University.
- KOZOVITS, L.E. 2008. *Mercado de Jogos para Celulares* [online] Active Tecnologia. Disponível em: <http://www.activetecnologia.com.br/j2me/MercadoJogosCelulares.pdf> [Acessado em 15 de Agosto de 2008].
- SHOLLBALL, 2008. *Showball Game* [online] Disponível em: www.jogos.etc.br/showball [Acessado em 15 de Agosto de 2008].
- SINGHAL, K.S., CHERITON, D., 1994. USING A POSITION HISTORY-BASED PROTOCOL FOR DISTRIBUTED OBJECT VISUALIZATION. TECHNICAL REPORT, DEPT. OF COMPUTER SCIENCE, STANFORD UNIVERSITY.
- SINGHAL, S., ZYDA, M., 1999. NETWORKED VIRTUAL ENVIRONMENTS: DESIGN AND IMPLEMENTATION. BOSTON:ADDISON WESLEY.
- SZWARMAN, D. FEIJÓ, B. AND COSTA, M., 2000. A framework for networked reactive characters, Proceedings of SIBGRAPI 2000, Gramado, Brasil, p. 203-210.
- UNITED STATES DEPARTMENT OF DEFENCE, 2002. Defence Modeling and Simulation Office. High Level Architecture. [online] Disponível em: <https://www.dms.mil/public/transition/hla/>. [Acessado em 6 de Agosto de 2007].

Dynamic Game Object Component System for Mutable Behavior Characters

José Ricardo S. Junior Erick B. Passos Esteban W. Clua Bruno C. Moreira Pedro C. Mourão

Instituto de Computação
Universidade Federal Fluminense

Abstract

Most games today use some form of Game Object Component System to compose game entities. With this approach, components represent anything such as functionalities or just a collection of attributes, and are attached to game objects in order to properly compose it. In this paper we propose an augmented Game Object Component System with automatic activation/deactivation of components based on runtime evaluation of logical conditions. Using this approach, it is possible to compose entities with mutable behavior based on such dynamically activated components. We propose this architecture as an alternative to Finite State Machines to model NPC behavior. This dynamic mechanism decouples the implementation of the behavior itself from its activation and deactivation, providing for easy reuse of such components in different game types by only modifying the activation rules.

Keywords: Dynamic component system, mutable behavior characters, engine architecture, finite state machine.

Authors' contact:

{joserikardo.jr,erickpassos}@gmail.com
pedrothiago@hotmail.com
brucostam2004@yahoo.com.br
esteban@ic.uff.br

1. Introduction

In game development, several techniques can be used to implement the behavior of game objects that are not controlled by the player. One example of these techniques are Finite State Machines (FSM), where a finite number of states acts as a memory of what happened in the past to make possible future decisions according to distinct events [WAGNER et al. 2006].

Using this approach, the game objects' behavior is specific for each state and transition rules have to be manually coded so that state switches happen throughout the simulation, providing for the mutable behavior.

However, to design a believable NPC behavior, many states have to be created. In this case, the management and maintenance of such architecture is a complex task, which is only advisable in cases where all the states are previously known [BAILLIE 2004].

In this paper we propose the extension of a Game Object Component System [BILLAS 2002; FOLMER 2007; STOY 2006] to develop Finite State Machines. In this new architecture, behavior components associated with a game object can be activated and deactivated through the dynamic verification of logical conditions. These logical conditions are composed with any number of game object attributes that are evaluated at runtime. Some novel features provided by this system regarding the design of game entities behavior are:

- Automatic dynamic mechanism for the activation of components (behaviors) through the use of logical conditions composed with runtime evaluation of game object attributes;
- Complete decoupling of components activation and implementation, providing for easier maintainability and reuse;
- Possibility to activate multiple/concurrent behaviors at the same time;
- Insertion and removal of new behaviors without increasing the code complexity of the others;

The rest of the paper is organized as follows: section 2 discusses related work, section 3 presents the concepts and design of the game object component system, and section 4 explains the dynamic mechanism for activation and deactivation of behaviors. Section 5 brings a case study of a Finite State Machine designed for a simple RPG NPC. Finally, section 6 concludes the paper and outlines future work.

2. Related Work

As the necessity for more realistic behavior of non player characters (NPC) in games increased, so did the number of states used in finite state machines used for this purpose. To tackle this issue, some architectures can be used to better manage and organize these states. One of them is called Hierarchical Finite State Machine (HFSM) [GIRAULT et al. 1999] where a set of states could be grouped together inside a super-state with some small states inside it. Using this approach, called generalized transitions, redundant transitions could be prevented. However, reusing transitions is not a trivial task and requires a lot of reasoning about the logic of several different contexts.

A technique called Object Oriented Hierarchical State Machine (OOHSM) [MALLOUK and CLUA, 2006]

was developed to better organize finite state machine. In this work, the author proposes the use of an abstract class which every concrete state machine needs to inherit from.

Our work decouples behaviors activation from the representation of the state machine itself, using object-oriented composition for the later and independent activation rules for the first. In the next section we explain our framework architecture and the dynamic activation mechanism for behavior components.

3. Game Objects Architecture

This section presents *GCore* [PASSOS et al. 2008], a game object component game framework implemented in Java, based on JMonkeyEngine [JMonkeyEngine], that has been developed as a platform for our research in game development and game software engineering.

3.1. The Game Object Class

In the proposed architecture, the *GameObject* class has attributes that are common for all objects in a game. Additionally, an object must have a set of components that are attached to it add new functionalities to it.

To enable communication between objects, there is a class responsible to manage all *GameObject* instances created in the game, where each object has its own unique identifier during game execution.

The types of game objects that are available in the game are stored externally in XML files. An example of how such definition could be is shows in code 1.

```
<types>
  <type name="basic">
    <component class="VisualComponent" />
  </type>

  <type name="npc-type" extends="basic">
    <component class="AIComponent" />
  </type>

  <type name="tree-type" extends="basic">
    <component class="VisualComponent">
      <model value="tree.3ds" />
    </component>
  </type>
</types>
```

Code 1: XML game type definition

In this example, a "BASIC-TYPE" is defined, specifying that all its instances will have a *VisualComponent* attached to them. This elementary type is referenced twice to illustrate two possible extension mechanisms: a) create new types by adding more components to the supertype, as shown in the example by the "NPC-TYPE", and b) specializing a type by modifying attributes of existing components.

3.2. The Base Component Class

In this architecture, the *BaseComponent* abstract class is the prototype of the behaviors where the logic of entities is defined. To make new components/behaviors, this class has to be extended, overriding the update method, according to the necessary functionalities. This class has a very important boolean attribute (enabled) to tell the *GCore* framework if each component instance is currently active. One should disable a component to avoid its update in the moment of the container game object's update. The *Template Method* design pattern [GAMMA et al. 1995] was used to implement this dynamic form of update.

Since some components depend on others to be executed properly, one should have a way to guarantee that this dependency will be satisfied in the game objects at runtime. To effectively tackle this issue, a technique called Dependency Injection [PASSOS et al. 2008] is used in our framework.

4. Dynamic Component Activation

Several benefits are obtained with rule-based dynamically activated game objects components. For instance, consider a game in which there is a game object type represents a student. Also, suppose this student could dynamically gain more abilities during the execution of the game. One example of such mutable behavior could be: if this student gets approved in all disciplines he becomes a doctor and gains the ability of doing medicine. Generally speaking, each new ability is represented by a specific component/behavior that should be activated to make the correspondent ability available, thus enabling the game object to practice medicine. Similarly, the deactivation of a component makes the container game object loose the correspondent ability. As long as the different behaviors are implemented independently, they can be very flexible and reusable in several games.

However, the dynamic management of behavior activations in such componentized FSM is a complex task. Doing this management manually is very difficult and error-prone, sometimes leading to unexpected results. A mechanism to do this task automatically can avoid many problems related to game prototyping and code maintenance. Our system aims to provide such mechanism, decoupling the implementation of behaviors from their activation, which is performed according rules optionally attached to each component at the game object types XML specifications.

Each component attached to a game object can have its own activation rule. This rule is a composition of logical expressions that is constantly evaluated at runtime. In case this rule evaluates to true to a specific component during game execution, this component is enabled, only for this particular instance of *GameObject*, and his update code is now executed at each game loop step. This dynamic evaluation is performed inside the *BaseComponent:Evaluate* (Template Method), which guarantees that the

overridden (user-defined) *Update* will be called only after the rule evaluates to true. This *Evaluate* method cannot be overridden and it is the one the *GameObject.Update* method actually calls.

The optional automatic activation condition for a component is defined as an expression composed of a binary logical operator and dynamically evaluated operands. Operands of an expression can be:

- Static values;
- A component attribute;
- Other expression.

Some of the available logical operators are:

- And (&);
- Or (!);
- Equals (=);
- Different (!=);
- Lower (<);
- Greater (>);

As will be shown in the next sections, expressions can be as complex as needed, and to enable their dynamic evaluation, component attributes values need to be obtained at runtime which is achieved by the use of Reflection. For instance, to make a component activation dynamic, based on attributes, an expression as the one shown in Code 2 will be needed. The `<activation>` tag is the container for the activation expression.

```
<type name="student-type">
  <component class="StudyInfo"/>
  <component class="DoctorBehavior">
    <activation>
      (StudyInfo.completed = true)
    </activation>
  </component>
</type>
```

Code 02: Attribute-based activation rule

In this example, the “student-type” game object type definition is composed of two components: *StudyInfo* and *DoctorBehavior*. The *StudyInfo* component does not have the `<activation>` tag, thus being always enabled. The `<activation>` tag included in the *DoctorBehavior* component holds the expression “(study-state.completed = true)”, which is parsed at load-time to a dynamic evaluator for the *Equals* logic operator.

From this explanation, it becomes clear that the *DoctorBehavior* of any “student-type” instance will be activated at runtime when the value of its *StudyState* component’s attribute named *completed* evaluates to the Boolean value *true*.

It is also possible to reference attributes from components of other game objects, as well as common available ones in the scene graph structure of the GCore framework. To specify another game object’s component attribute, one has to include this game

object identifier in the operand specification. In this case, both operands can be dynamically evaluated attributes, as can be seen in Code 3.

```
<type name="a-type" extends="basic">
  <component class="Behavior"/>
</type>

<type name="b-type" extends="a-type">
  <component class="Behavior">
    <activation>
      ( (comp1.attributeA != 1)
        &
        (comp1.attribute2 = 473) )
    </activation>
  </component>
</type>
```

Code 03: Decoupled activation

In this example, game objects of type “a-type” will always have their Behavior components activated, while instances of “b-type” will depend on the runtime evaluation of the logical expression to do so. The code also shows that expressions have to be expressed in their complete parentized form, as illustrated by the condition composed with “&”, “!=” and “=” operators.

With the architecture shown in this section, complex NPC behaviors can be broken in several discrete components that will be automatically managed using dynamic activation and deactivation.

5. Case Study: NPC Simulation

To show how this approach can be used in a real game scenario, an exclusive prototype was developed. This prototype presents a land infested with zombies, which like to “kill” each other. The zombies that are put in the scene interact with each other through dynamic activated mutable behaviors. To achieve the desired result, a simple *ZombieState* component was created to hold common referenced attributes. This component is responsible to gather and cache all information that is needed by the dynamic behaviors such as the zombie’s health, nearest zombie (enemy) and the distance to it.

For the zombies to interact with each other, some behaviors were developed and added to them. The available behaviors and the desired activation rules for them are listed in Table 1.

To use such behavior activations with a Finite State Machine, one has to discover what are the possible states represented by them. One way of doing this is to construct a truth-table for all the conditions expressed at the activation rules, remembering to evaluate only the ones that are independent. Also notice that with the above definitions, more than one behavior could be active at the same time. Table 2 shows how even such simple example increases the complexity of an equivalent FSM, as number of states is directly dependent on how many independent activation rules exist.

Based on the complete activation rules given in Table 1, it is possible to realize that some condition

combinations lead to the activation of the same behaviors, so the actual Finite State Machine that needed to be specified would have only 5 states.

Table 1: Behavior and their activation rules

Behaviors	Activation Rule
Wander	state.enemyDistance $\geq$ 100
Pursue	state.enemyDistance $\geq$ 20 & state.enemyDistance < 100 & state.health $\geq$ 40
Shot	state.health $\geq$ 40
Evade	state.health < 40 & state.enemyDistance < 100

Table 2: Truth-table of possible states

State\Condition	Distance < 100	Distance $\geq$ 20	Health $\geq$ 40
State 1	T	T	T
State 2	T	T	F
State 3	T	F	T
State 4	T	F	F
State 5	F	T	T
State 6	F	T	F
State 7	F	F	T
State 8	F	F	F

Even in this simple case, the programmer would have to implement, besides the behaviors, the FSM manager component and the transition evaluations and consequent behavior activations. Our automatic mechanism largely simplifies this task, and as more complex the FSM becomes, the more difficult the manual implementation is, eventually leading to maintainability issues.

6. Conclusion

Data driven game object composition is a proved approach to put down risks in game development. Such flexible architecture has been used in several successful engines, frameworks and games. However, the design and implementation of game entities that need mutable behavior brings some issues related to maintainability and code reuse. In this paper, we presented a game framework with a dynamic activation mechanism for components that can be used to

implement complex characters. This feature provides for the decoupling of behavior implementation and its activation, which permits behaviors to be seamlessly reused with other entity types, even when its activation rules (state machine) are different.

The *GCore* framework is a constant work in progress and we are now investigating how use different activation mechanisms for the behaviors such as scene triggers and other in-game events. We also plan to extend this to include timed behavior activation, where it will be possible to also specify for how long should last the activation. This will facilitate the design of temporary behaviors, which are useful in some situations in game development. The short-term roadmap also includes a level-editor, enabling game objects behaviors composition and activation rules editing in visual form.

References

- BAILLIE, P., 2004. *Programming Believable Characters in Computer Games*, Massachusetts: Charles River Media, Inc.
- BILLAS, S., 2002. A data-driven game object system. Talk at the Game Developers Conference '02.
- FOLMER, E. 2007. Component based game development – a solution to escalating costs and expanding deadlines? *In Component Based Software Engineering*, Springer, vol. 4608 of Lecture Notes in Computer Science, 66-73.
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.
- GIRAULT, A., BILUNG, L., LEE, E., 1999. Hierarchical finite state machines with multiple concurrency models. *In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 06 August 1999 Berkeley, California: IEEE Press, 742-760.
- JMONKEYENGINE. Jmonkey engine 1.0 online documentation. <http://www.jmonkeyengine.com/>.
- MALLOUK, W., CLUA, E., 2006. An Object-Oriented Approach for Hierarchical State Machines. *In: Proceedings of the SBGames conference in Computing*, 8-10 November 2006 Recife, Brazil.
- PASSOS, E., SILVA, J., CLUA, E., 2008. Fast and Safe Prototyping of Game Objects with Dependency Injection. Tech Report. Universidade Federal Fluminense -- UFF. 2008. <http://www.ic.uff.br/PosGraduacao/RelTecnico/385.pdf>
- STOY, C., 2006. Game object component system. *In: Game Programming Gems 6*. Charles River Media, M. Dickheiser, Ed., 393-403.
- Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P., 2006. *Modeling software with Finite State Machine: A practical approach*, United States: Taylor & Francis Group.

Move-In: Uma API para Interação com Webcam

Jeferson José de Miranda*

Marcelo da Silva Hounsell

Alexandre Gonçalves Silva

LARVA – Laboratório de Realidade Virtual Aplicada, DCC – Departamento de Ciência da Computação, UDESC – Universidade do Estado de Santa Catarina, Joinville, SC, Brasil

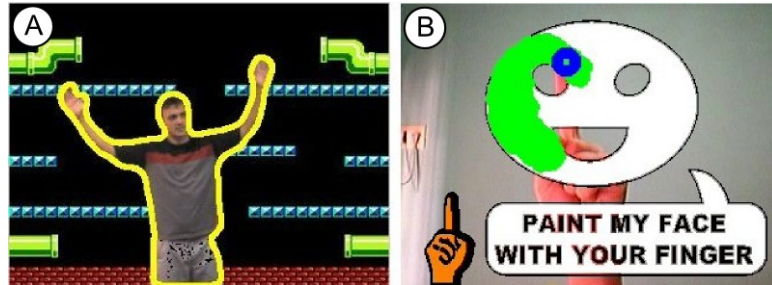


Figura 1: Aplicações da API Move-In.

Resumo

Este artigo descreve a implementação de uma API, denominada Move-In, de código aberto criada para facilitar o desenvolvimento de aplicações onde a comunicação com o *software* se dá através do movimento do corpo do usuário capturado com uma *webcam*, sem o uso de marcadores. As funcionalidades resultantes tornam possível a identificação da presença e do movimento de um usuário em frente a uma *webcam*; a interação com objetos virtuais e; a identificação de certas posturas do corpo e da mão humana. Testes mostraram que a API pode ser usada para prototipação de aplicações com captura de vídeo.

Palavras-chave: processamento de imagens, visão computacional, captura de vídeo, captura de movimento.

Abstract

This paper describes the implementation of an open source API, named Move-In, created to ease the development of applications where the communication with the software is given by the users' body motion captured with a webcam without markers. The resulting functionalities make it possible to identify user's presence and movements in front of the webcam; the interaction with virtual objects and; the identification of some human hand or body postures. Tests have shown that the API can be used for prototyping video captured applications.

Keywords: image processing, computer vision, video capture, motion capture.

Authors' contact:

*jeferson.miranda@gmail.com
{marcelo,alexandre}@joinville.udesc.br

1. Introdução

Uma forma de facilitar a importação de recursos de processamento de imagens e visão por computador para ter uma mais fácil integração a jogos, sem que o projetista tenha que se tornar um especialista, está se tornando cada vez mais necessária. Neste contexto, este trabalho apresenta a API Move-In que tem o objetivo de facilitar a aquisição da imagem de entrada, a segmentação da área de interesse e a extração das características. A análise das características é de responsabilidade do projetista do *software* que fará interação com a *webcam*.

A API Move-In implementa quatro técnicas de identificação de movimento ou de presença e uma forma de gerenciar a região da imagem de entrada afetada por essas técnicas. Dentro do contexto deste trabalho, a presença deve ser considerada como a detecção do usuário (parado) em frente à câmera. Já o movimento se refere à detecção de mudanças de posição entre uma sequência de imagens.

A imagem de entrada da API são os quadros capturados pela câmera. Pode-se definir N regiões de interesse (*Region Of Interest* – ROI) para partes específicas da imagem, como um botão virtual no canto superior esquerdo ou um objeto virtual que se movimenta pelo centro da tela. Na API Move-In as ROIs são espaços retangulares não orientados. Isso facilita a implementação, otimiza a execução e resulta em dados equivalentes à implementação de regiões com formato arbitrário. Fica a cargo do programador definir o conjunto de ROIs que será associado ao objeto virtual.

2. Implementação

O conjunto de funcionalidades, que constitui o resultado deste trabalho, foi implementada sobre a API

OpenCV [Intel 2006], escolhida pela sua eficiência, número de algoritmos de processamento de imagens e visão computacional implementados e pela possibilidade de utilização com *software* comercial.

A API Move-In é um conjunto de classes codificadas em C++ que foi construída utilizando como base as funções estruturadas da versão 1.0 da API OpenCV. A API Move-In está organizada em quatro conjuntos de classes, ou módulos do sistema: **Classes auxiliares**: constituem abstrações para o paradigma orientado a objetos de funcionalidades do OpenCV referentes à criação de imagens e conexão com *webcams*; **Estruturas de dados**: agrupa as classes criadas para armazenar os dados referentes às características extraídas das imagens; **Gerenciamento de regiões de interesse (ROI)**: gerencia as regiões retangulares que limitam o cálculo de segmentação da área de interesse a uma região específica da imagem capturada pela câmera; **Classes do núcleo do sistema**: agrupa as classes responsáveis pela segmentação da área de interesse e extração das características (objetivo deste trabalho).

Existe uma classe, a classe Screen, que é a única que não faz parte de um dos módulos do sistema pelo fato de que ela centraliza o acesso às principais funcionalidades. Isso atende ao padrão de projeto (*design pattern*) *facade* [Gamma et al. 1995].

Uma das características que deve ser observada na hora da escolha da webcam para uso com a técnica de presença na API Move-In é a possibilidade de desabilitar o ganho automático dos níveis de branco. Isso porque o resultado da diferença entre quadros é efetuado a partir das variações de intensidade dos valores RGB, e a mudança brusca na configuração do cenário resulta em grandes diferenças de intensidade de iluminação entre o quadro que representa o cenário de fundo e o último quadro capturado.

2.1 Formas de Segmentação

A API Move-In utiliza duas formas de segmentação da área de interesse, a saber: (1) diferença entre quadros de animação e (2) identificação da cor da pele. A partir da diferença entre quadros pode-se obter dados de presença ou movimento. A Figura 2 expõe as formas de segmentação do tipo 1.

A Figura 2 mostra, respectivamente, o quadro inicial capturado com evento de teclado e contendo somente o cenário de fundo (Figura 2.A), o quadro N-1 (Figura 2.B), o quadro N (Figura 2.C), a diferença entre N e N-1 (Figura 2.D), a diferença entre o quadro inicial e o quadro N no espaço de cor RGB (Figura 2.E), e essa mesma diferença no espaço de cor RGB normalizado (Figura 2.F). A primeira segmentação (Figura 2.D) identifica movimento e as duas últimas (figuras 2.E e 2.F), presença.

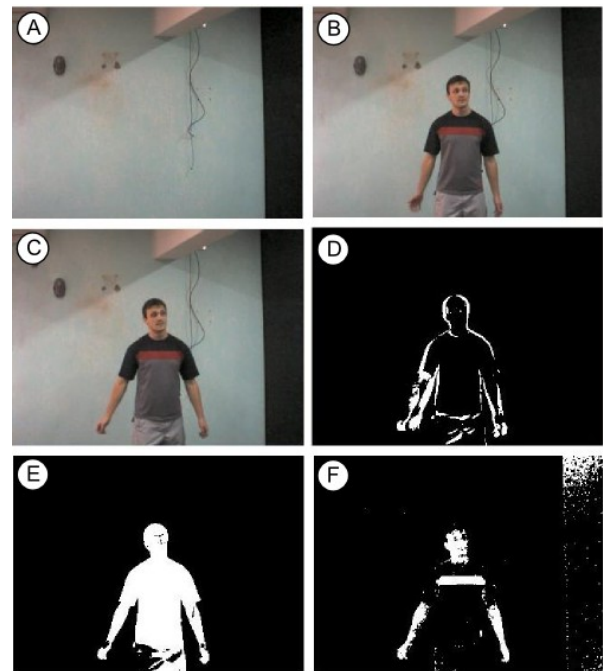


Figura 2: Formas de segmentação da área de interesse.

A segmentação obtida a partir da diferença entre quadros depende do ajuste de um limiar ou fator de corte. As intensidades de nível de cinza resultantes da imagem subtraída variam de 0 a 255. Na binarização dessa imagem, valores menores ou iguais que o limiar são tratados como branco (255) e valores acima do limiar são tratados como preto (0).

Para as segmentações do tipo presença, que objetivam identificar toda a silhueta do usuário parado observou-se, a partir de experiências realizadas de forma empírica, que os limiares que devem ser utilizados para o espaço de cor RGB, sob pena de produzir muitos falsos positivos ou falsos negativos, variam entre 20 e 50, e para o espaço de cor RGB normalizado variam de 5 a 20. Isso porque a normalização das cores diminui as variações bruscas de intensidade entre os *pixels*. Um limiar abaixo de 20 para o espaço de cor RGB tende a ser extremamente sensível à mudanças de iluminação durante a sequência de imagens.

A Figura 2 mostra que a segmentação por diferença de quadros do tipo presença dificilmente produz um resultado ótimo, tanto para o espaço de cor RGB quanto para RGB normalizado. No entanto, esses resultados podem ser combinados, e muitos falsos positivos podem ser eliminados.

A partir deste experimento pode-se concluir que a segmentação por diferença de quadros do tipo presença é extremamente dependente dos valores de limiar e do pós-processamento efetuado a partir dos resultados da segmentação. Já o mesmo não ocorre para a diferença entre quadros do tipo movimento, que geralmente não depende de ótimos resultados para as aplicações as

quais se destina, e pode ser resolvido no espaço de cor RGB com limiares que variam entre 40 e 60.

A Figura 3 mostra o segundo tipo de segmentação obtido pela identificação da cor da pele. A figura mostra a imagem original (Figura 3.A) seguida da segmentação nos espaços de cor RGB (Figura 3.B) e RGB normalizado (Figura 3.C).

Na API Move-In, a cor da pele é segmentada a partir de uma equação paramétrica no espaço de cor RGB definida no trabalho de Gasparini e Schettini [2006]. Esta equação foi obtida de forma empírica e avalia os valores dos componentes RGB, classificando como cor da pele aqueles que se encontram dentro de um determinado intervalo.



Figura 3: Segmentação da cor da pele.

A imagem binária resultante da segmentação da cor da pele produziu muitos falsos positivos e demonstra que o algoritmo não é ótimo. Por esse motivo, esse tipo de segmentação deve ser utilizado em casos onde não há elementos no cenário, além da própria pele do usuário, que façam parte do intervalo de valores definido como pele pela equação paramétrica (como no exemplo da Figura 6).

2.2 Gerenciamento de Regiões de Interesse

O não uso de toda a imagem para realizar uma das formas de segmentação é freqüente. A utilização de regiões de interesse (ROIs) ocorre principalmente na ativação de botões virtuais e movimentação de objetos virtuais. Para tanto, a API Move-In conta a classe ROIGroup, onde um conjunto de regiões de interesse pode ser criado para uso conjunto com as técnicas de segmentação.

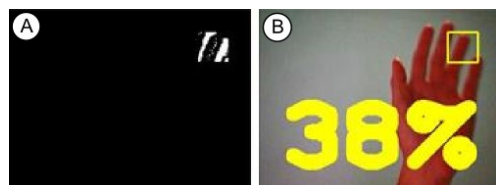


Figura 4: Demonstração de ativação de botão virtual.

A Figura 4 demonstra como uma ROI pode ser usada em conjunto com a segmentação do tipo movimento para ativação de botão virtual. Para a ROI localizada no canto superior direito da Figura 4.B, a diferença entre os dois últimos quadros é realizada Figura 4.A. Após isso, calcula-se a porcentagem de ativação, ou proporção de *pixels* brancos dentro da ROI (38% na Figura 4.B). Dessa forma, um botão pode

ser programado para ser ativado após uma sequência de quadros cuja porcentagem de ativação seja superior a 25%, por exemplo.

Se um conjunto de ROIs for utilizado em volta de um objeto virtual, conforme Figura 5, pode-se permitir que o usuário o “empurre” ao longo da tela. Isto é, movimentos detectados na ROI 0 deslocam o objeto, e, conseqüentemente, a ROIGroup, para a direita.



Figura 5: Organização de ROIs para movimentação de objeto virtual.

Movimentos detectados nas ROIs 1, 2 e 3, respectivamente, deslocam o objeto virtual para baixo, direita e esquerda. Experimentos realizados concluíram que 4 ROIs, conforme demonstrado na Figura 5, são suficientes para esta função. Complementarmente, pode-se compor movimentos caso sejam detectados duas ROIs ao mesmo tempo, gerando movimentos inclinados, por exemplo.

2.3 Extração de Características

As características que podem ser extraídas a partir de uma imagem segmentada são: **Área ativa**: proporção dos pixels brancos dentro de uma ROI; **Centróide**: ponto médio aproximado da região afetada; **Contorno**: sequência de pontos referente ao contorno dos *blobs*; **Fecho convexo**: o fecho convexo referente ao contorno; **Pontos extremos**: o fecho convexo acrescido dos defeitos de convexidade, que consiste de um ponto côncavo que maximiza a distância de uma região côncava entre dois pontos convexos.

Exemplo de uso da característica “área ativa” aparece na Figura 4 (botão virtual). O “contorno” pode ser utilizado para destacar os limites entre o usuário real e o cenário virtual (figura 1.A). O uso dos “pontos extremos” pode ser conferida na Figura 6.

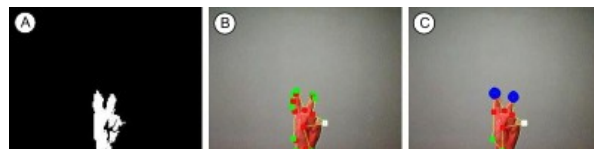


Figura 6: Análise da característica “pontos extremos”.

Na Figura 6.A, os pontos extremos foram calculados a partir da imagem segmentada com identificação da cor da pele. O pontos convexos (em verde na Figura 6.B) foram eliminados e substituídos pela média entre esses pontos quando estes se encontravam muito próximos. Então, as seguintes

heurísticas (análise de características) foram aplicadas para determinar quais dedos estavam erguidos:

1. O ponto i , sendo i sua posição na lista de pontos, deve ser convexo;
2. Os pontos $i-1$ e $i+1$ devem ser côncavos;
3. O ponto i deve se encontrar em uma posição (altura) superior à dos pontos $i-1$ e $i+1$;

O resultado aparece em azul na Figura 6.C. Com a ponta do dedo, a aplicação de pintura com o dedo da figura 1.B foi criada.

2.4 Fluxo Óptico

A utilização da classe Flow da API Move-In está ilustrada na Figura 7. Primeiramente, o conjunto de pontos a serem rastreados é determinado (Figura 7.A). Após isso, os vetores de movimentação referentes a esses pontos são calculados (Figura 7.B). Nas figuras 7.B e 13.C os vetores foram aumentados em um fator de 10X para facilitar a visualização.

A Figura 7.C mostra o vetor resultante calculado a partir dos vetores ilustrados na Figura 7.B. A soma dos vetores dá a direção do vetor resultante, e sua magnitude é calculada pela média aritmética das magnitudes dos conjuntos totais de vetores.

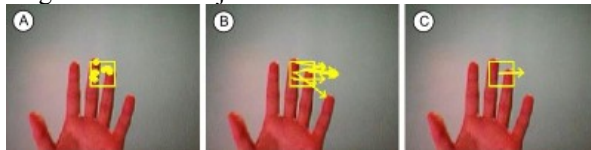


Figura 7: Pontos rastreados, vetores de movimentação e vetor resultante do fluxo óptico.

É o projetista da aplicação que define a quantidade de pontos a serem rastreados. O vetor resultante é utilizado opcionalmente, mas geralmente é indispensável, já que é ele quem determina a direção e velocidade do movimento.

O cálculo dos pontos a rastrear é efetuado automaticamente por uma função da API OpenCV. Alternativamente, pode-se definir quais pontos devem ser rastreados. O resultado é o rastreamento de um objeto real ao invés da leitura de movimento sobre um objeto virtual. Esta alternativa é recomendada somente para objetos deslocados nas duas dimensões correspondentes à largura e altura da imagem de entrada, sob pena de apresentar resultado extremamente impreciso.

As aplicações do fluxo óptico variam como forma alternativa à utilização da segmentação do tipo movimento. O fluxo óptico tem a capacidade de efetuar a movimentação de um objeto virtual (Figura 5) com a utilização de uma única ROI.

O cálculo do fluxo óptico é muito sensível a quaisquer mudanças de iluminação em uma sequência de imagens. Dessa forma, seu uso é recomendado em

conjunto com a detecção de movimento com diferença entre quadros.

3. Conclusão

Este trabalho apresentou os detalhes do projeto, implementação e funcionamento da API Move-In, que visa facilitar o desenvolvimento de aplicações interativas, onde o usuário usa o próprio corpo, identificado por uma *webcam*, durante a comunicação com a aplicação.

A API Move-In criou uma camada de abstração sobre a API OpenCV que, além de agrupar um conjunto de funcionalidades relacionadas à identificação de presença e movimento do usuário, ainda provê formas de gerenciamento dessas funcionalidades em regiões específicas definidas pelo projetista da aplicação. A API Move-In está disponível no *website* <http://code.google.com/p/move-in/> e maiores informações sobre esta pesquisa podem ser obtidas em <http://www.joinville.udesc.br/larva>.

Neste trabalho ficou evidente a relação de dependência entre a detecção de movimentos rápidos e a qualidade da *webcam* em uso somada à capacidade de processamento dos computadores. Jogos que simulam esportes como tênis de mesa ou voleibol, dificilmente serão aplicados com sucesso com o *hardware* disponível na maioria das residências da atualidade, a não ser que o projetista limite a velocidade de movimentação da bola para estimular o jogador a não utilizar movimentos muito bruscos.

Foi observado que o trabalho com técnicas de identificação de movimento é recomendado para uso doméstico quando em comparação às técnicas de identificação de presença devido ao maior grau de robustez em relação a mudanças de iluminação. Por outro lado, identificou-se que a identificação de poses ou posturas do corpo e das mãos não é possível somente com técnicas de movimento.

Referências

- GAMMA E., HELM R., JOHNSON R., VLISSIDES J., 1995. Design Patterns: Elements of Reusable Software. Addison-Wesley.
- GASPARINI, F., SCHETTINI, R., 2006. "Skin Segmentation Using Multiple Thresholding". In: *Proceedings of SPIE - The International Society for Optical Engineering*, 15-19 January 2006 San Jose.
- INTEL., 2006. *Open Source Computer Vision Library* [online]. Disponível em: <http://www.intel.com/technology/computing/opencv/> [Acessado em 13 de agosto 2008].
- SONY COMPUTER ENTERTAINMENT INC., 2003. *Sony Eye Toy* [online]. Disponível em: <http://www.eyetoy.com/index.asp> [Acessado em 13 de agosto de 2008].

Algoritmo para Verificação da Solução de Quebra-cabeças Geométricos

Manoel Siqueira Junior
Clayton Reis da Silva

Rafael Alves
Anselmo Montenegro

Wagner dos Santos
Erick Passos

Carol Carvalho
Esteban Clua

UFF MediaLab

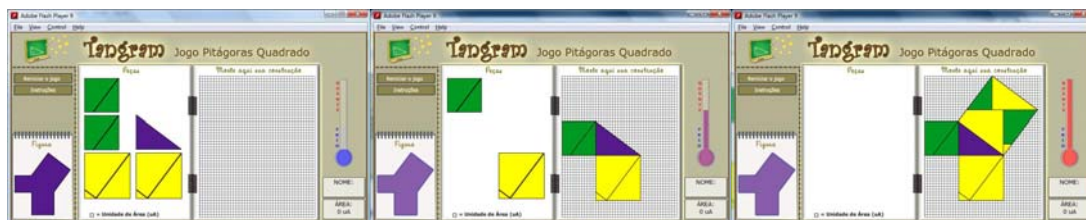


Figura 1: Jogo de Tangram demonstrando, por meio de um termômetro, quão perto da solução o jogador está

Resumo

Nesse trabalho, apresentamos um algoritmo que resolve o problema de verificação da solução de um quebra-cabeças geométrico. São classificadas dezesseis possíveis relações entre as arestas de cada polígono de forma a eliminar aquelas que não façam parte do contorno. Esse método permite a verificação de semelhança entre uma figura modelo e uma dada montagem sem a necessidade de meta-informações sobre a solução.

Abstract

In this paper, we present an algorithm to solve the problem of correctly verifying the solution of geometric puzzles. Sixteen possible relations between the original polygon edges are classified to eliminate those that are not part of the final figure. This method provides for the verification that an arranged set of polygons forms the same image as the desired solution without the need of extra meta-data but the vertexes themselves.

Palavras-chave: quebra-cabeças geométrico, contorno de polígono, tangram

Contato dos autores:

manoeljr88@gmail.com
rafamachadoalves@yahoo.com.br
rengaw.lui@gmail.com
carol@carolcruz.net
{creis,anselmo,epassos,esteban}@ic.uff.br

1. Introdução

O algoritmo que será discutido nesse trabalho é uma solução alternativa para o problema da identificação se uma determinada montagem de peças de um quebra-cabeças representa a figura desejada. Tomou-se como base para a elaboração dessa solução o algoritmo do

apresentado por Scarlatos [SCARLATOS 1999], que deixa margem a trabalhos futuros, pois não é adequada para quebra-cabeças que possuem mais de uma montagem possível.

O algoritmo proposto usa uma abordagem diferente, de forma que seja possível se verificar arranjos distintos de peças que levem à mesma figura, considerando apenas o contorno que é formado pelas peças unidas, não o modo como elas estão dispostas dentro desse contorno, além de analisar se a montagem final não possui furos.

É importante ressaltar que cada peça do quebra-cabeça é representada por um polígono, assim como a figura modelo. O restante do trabalho é organizado da seguinte forma: a seção 2 apresenta alguns trabalhos relacionados à implementação e métodos de verificação de relações entre arestas em figuras poligonais. A seção 3 descreve os tipos de quebra-cabeças que são tratados pelo método proposto, enquanto a seção 4 faz uma análise dos problemas a serem tratados. A seção 5 explica o algoritmo desenvolvido. Finalmente, a conclusão é apresentada na seção 6. No final do artigo, estão incluídas tabelas úteis, referenciadas ao longo do texto, cujos tamanhos tornaram inviável sua disposição ao longo do mesmo.

2. Trabalhos Relacionados

Nessa seção, buscamos comparar a nossa técnica com outras implementações de quebra-cabeças matemáticos pesquisadas, especificamente em relação ao reconhecimento da solução, além de trabalhos sobre métodos genéricos de reconhecimento de figuras através da classificação de relações entre arestas.

Como exemplo de um quebra-cabeça geométrico, tem-se o Tangram ZTOR [ZTOR 2005], que faz o reconhecimento da solução de forma bastante simplificada, não permitindo a montagem em rotações alternativas. Além disso, o mesmo não faz o

reconhecimento parcial da solução, o que possibilita o usuário saber, em escala de zero a cem por cento, o quão perto ele está de encontrar a solução final do desafio. Nossa solução permite tanto a rotação arbitrária quanto indica a solução parcial.

Em [FLASHKIT 1999], pode-se encontrar um ótimo repositório de códigos fonte, tutoriais e outros materiais para desenvolvimento de jogos em Flash, incluindo muitos exemplos de Tangrams ([LANKIN 2001], [JACOB 2002], [MARTINS 2002] e [Texas 2000]). Entretanto, nenhum desses exemplos apresenta algoritmos para verificação de solução, servindo apenas como referência para implementação da mecânica básica.

Em [SCARLATOS 1999] é demonstrado um método para se reconhecer uma figura formada por diversos polígonos justapostos através das relações exatas formadas por suas arestas. Essa solução, entretanto, só reconhece cada possível solução individualmente. Uma figura que possa ser formada por mais de uma combinação de peças possui múltiplas representações, o que torna inviável sua utilização para Tangrams, que podem ter milhares de possíveis soluções até para instâncias simples. Mesmo não sendo satisfatório, o referido trabalho serviu como base para o método desenvolvido e mais detalhes serão apresentados na seção 4.

Uma situação na qual arranjos diferentes de polígonos formam a mesma figura é ilustrada na figura 1. Nesse exemplo, a sub-figura quadrada formada pela união dos polígonos 2 e 3 pode ser encaixada com a figura 2 em qualquer rotação. Enquanto o método referenciado [SCARLATOS 1999] gera duas representações diferentes, o mecanismo que está sendo proposto nesse artigo reconhece ambas como figuras iguais, sendo então mais apropriado para o problema da verificação de soluções de quebra-cabeças geométricos.

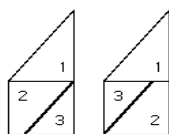


Figura 2: Figura montada de duas maneiras diferentes usando os mesmos polígonos

Até o presente momento, não foi encontrado nenhum trabalho que apresente um método genérico para o reconhecimento de solução de quebra-cabeças geométricos a partir da comparação com uma figura-exemplo. No restante do artigo, serão mais bem explicados os quebra-cabeças geométricos e o método desenvolvido.

3. Quebra-cabeças geométricos

Os quebra-cabeças geométricos são aqueles constituídos de peças representadas por polígonos, que se encaixam de tal forma que o contorno resultante, ou seja, a borda do conjunto de peças encaixadas, forme a

figura solução. Diferentemente dos quebra-cabeças tradicionais, que se baseiam no desenho estampado em cima de cada peça, as quais se encaixam entre si para formar determinada imagem, apenas esse contorno é importante na verificação de que foi atingida a solução.

4. Considerações sobre o problema

Para que o arranjo das peças no decorrer do jogo seja verificado, relações entre as arestas dos distintos polígonos, propostas em outro trabalho de pesquisa [SCARLATOS 1999], são consideradas para auxiliar na identificação dos contornos da figura atual montada pelo jogador. Além das relações básicas entre arestas, ajustes são realizados para que o resultado final dessa montagem seja comparado de maneira adequada com a figura solução.

Tanto a representação da solução quanto a montagem feita pelo jogador são feitas por listas circulares de arestas. É importante salientar que os polígonos formados durante o jogo podem conter furos e isso não interfere na análise da solução proposta pelo jogador.

5. Heurística de verificação

Essa heurística de verificação da solução de um quebra-cabeça geométrico utiliza das relações descritas em [SCARLATOS 1999], porém as trata de uma maneira diferente, de forma que sobrem apenas arestas de contorno. Depois são feitos ajustes para que a figura seja representada de uma única maneira.

5.1 Relações

As relações consideradas sempre são verificadas entre arestas de polígonos distintos. Supondo que uma aresta seja formada pelos pontos a_1 e a_2 e a outra, por b_1 e b_2 , o que caracteriza cada relação é o fato de cada um desses pontos pertencer ou não à outra aresta analisada. Se o ponto pertencer à outra aresta, será atribuído valor 1 a ele e, caso contrário, 0. Concatenando os valores de a_1 , a_2 , b_1 e b_2 , nessa ordem, teremos a representação da relação entre duas arestas expressa em forma binária, bastando apenas, caso seja conveniente, convertê-la para decimal.

Existem relações que dividem uma aresta em duas, as que são nulas e as que eliminam total ou parcialmente arestas. Essas relações serão descritas a seguir.

5.1.1 Relação de divisão

Esse tipo de relação ocorre quando apenas um dos vértices é tangente à aresta analisada ou, quando os dois vértices extremos de uma mesma aresta estão contidos em outra, não tocando nos pontos extremos dessa.

Tomando como base os pontos a_1 , a_2 , b_1 e b_2 , segue, na tabela 1, uma descrição de cada relação contida nesse

grupo.

5.1.2 Relação nula

Esse tipo de relação ocorre quando nenhum vértice da extremidade das arestas sendo verificadas toca na outra, ou quando a interseção entre essas duas arestas resulta apenas em um vértice em comum.

Analogamente à subseção anterior, segue, na tabela 2, uma descrição de cada relação contida nesse grupo.

5.1.3 Relação de eliminação

Esse tipo de relação ocorre quando os dois vértices da extremidade de uma das arestas tocam na outra aresta analisada. Também ocorre esse tipo de relação quando um dos vértices da extremidade de cada aresta toca na outra, mas esses pontos não possuem a mesma coordenada no plano cartesiano.

Analogamente às subseções anteriores, segue, na tabela 3, uma descrição de cada relação contida nesse grupo.

5.2 Ajuste do resultado gerado pelas relações

Após as relações serem aplicadas, arestas de contorno são encontradas, porém falta ajustá-las de forma que uma ou mais figuras finais sejam identificadas. Esse ajuste é feito selecionando uma aresta qualquer para ficar na posição inicial da lista circular utilizada para armazenar o polígono final formado e, após essa disposição inicial, basta-se posicionar as demais arestas na posição correta dessa estrutura de dados, a partir das relações verificadas, para que se tenha a representação da figura final. Neste momento também são unificadas arestas adjacentes que formem ângulo de 180° entre si. Essas operações são feitas até que todos os contornos sejam identificados.

Por meio desse método, os contornos são definidos, podendo ser de polígonos de furo, caso sejam desenhados no sentido inverso do predefinido pelas peças, ou, com área interna, caso sejam desenhados no outro sentido, como é observado na figura 5.

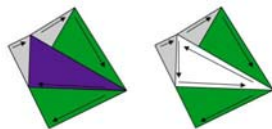


Figura 5: Diferenças entre um polígono sem e com furo

Para verificar se um polígono com área interna representa a solução, faz-se uma lista circular que armazena estruturas que possuem duas informações: o comprimento de uma aresta presente na lista circular que representa o polígono solução do jogador e o ângulo entre essa aresta e a próxima desta lista. Essas estruturas devem seguir a ordem da lista circular que representa a solução submetida pelo jogador. Esse mesmo tipo de lista circular é feito para a figura modelo (solução) do jogo.

Após estarem prontas essas listas com as estruturas descritas no parágrafo anterior, basta comparar a da solução do jogo com a do jogador. Se algum dos polígonos com área interna encontrados formarem a figura solução, o próximo passo será verificar a existência de algum polígono de furo dentro do mesmo.

Caso não exista furo na figura com área interna que possui contorno igual à solução, chega-se à conclusão de que o jogador montou a solução correta.

5.3 Percentual de acerto

Para o cálculo do percentual de acerto, obtém-se o mínimo entre a porcentagem do contorno dos polígonos montados pelo jogador que mais se aproxima da figura solução e a da área preenchida desse contorno. Essa medição de progresso possui mais uma motivação de usabilidade do que de precisão, uma vez que permite a percepção de progresso por parte do jogador.

6. Conclusão e trabalhos futuros

Neste artigo, apresentou-se uma nova abordagem sobre as relações entre arestas de polígonos adjacentes, de forma a se obter uma gama maior de casos em que a verificação das soluções para o problema da construção de quebra-cabeça seja feita de maneira correta. Essa proposta soluciona falhas de outras anteriores, que não levam em consideração apenas a forma final da solução montada pelo jogador, e sim toda a disposição de peças montadas.

Futuramente, vários incrementos deverão ser adicionados a esse algoritmo, como:

- Levar em consideração peças e soluções com fronteiras curvas;
- Abranger peças e soluções com furo;
- Adaptação do algoritmo para problema semelhante em três dimensões.

Agradecimentos

Esse projeto é financiado pelos Ministérios da Educação e da Ciência e Tecnologia através da chamada pública de desenvolvimento de conteúdo digital para o ensino médio.

Referências

- LANKIN, A., 2001. *Implementação de Tangram*. Disponível em: <http://www.flashkit.com/movies/Games/Full_Game_Soource/Tangram-Andrew_L-5966/index.php>. Acesso em: 03/08/2008.
- JACOB, E., 2002. *Implementação de Tangram*. Disponível em:

<http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Eduardo_-8137/index.php>.
Acesso em: 03/08/2008.

FLASHKIT, 1999. Seção: *Movies*; Subseção: *Games*.
Disponível em: <<http://www.flashkit.com>>.
Acesso em: 03/08/2008.

MARTINS, I. F., 2002. *Implementação de Tangram*.
Disponível em:
<http://www.flashkit.com/movies/Games/Tangram-Illcio_-6471/index.php>.
Acesso em: 03/08/2008.

JASP, T., 2000. *Implementação de Tangram*.
Disponível em:
<http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Texas_Ja-2061/index.php>.
Acesso em: 03/08/2008.

ZTOR, 2005. *Implementação de Tangram*.
Disponível em:
<<http://www.ztor.com/index.php4?ln=&g=game&d=tang>>.
Acesso em: 03/08/2008.

SCARLATOS, L. L., 1999. *Puzzle piece topology: detecting arrangements in smart objects interfaces*.

Relação	a_1	a_2	b_1	b_2	Descrição
1	0	0	0	1	Aresta formada pelos pontos a_1 e a_2 é subdividida pelo ponto b_2
2	0	0	1	0	Aresta formada pelos pontos a_1 e a_2 é subdividida pelo ponto b_1
3	0	0	1	1	Subdivide a aresta formada pelos pontos a_1 e a_2 em duas: uma cujos pontos extremos são a_1 e b_2 e outra, b_1 e a_2 , nessa ordem, pois assim o sentido das arestas é preservado
4	0	1	0	0	Aresta formada pelos pontos b_1 e b_2 é subdividida pelo ponto a_2
8	1	0	0	0	Aresta formada pelos pontos b_1 e b_2 é subdividida pelo ponto a_1
12	1	1	0	0	Subdivide a aresta formada pelos pontos b_1 e b_2 em duas: uma cujos pontos extremos são b_1 e a_2 e outra, a_1 e b_2 , nessa ordem, pois assim o sentido das arestas é preservado

Tabela 1: Descrição de relações de divisão de arestas

Relação	a_1	a_2	b_1	b_2	Descrição
0	0	0	0	0	Como se trata de arestas disjuntas, nada faz
5	0	1	0	1	Como apenas os pontos a_2 e b_2 se tocam, não havendo necessidade de alterar as duas arestas, nada faz
6	0	1	1	0	Como apenas os pontos a_2 e b_1 se tocam, não havendo necessidade de alterar as duas arestas, nada faz
9	1	0	0	1	Como apenas os pontos a_1 e b_2 se tocam, não havendo necessidade de alterar as duas arestas, nada faz
10	1	0	1	0	Como apenas os pontos a_1 e b_1 se tocam, não havendo necessidade de alterar as duas arestas, nada faz

Tabela 2: Descrição de relações nulas

Relação	a_1	a_2	b_1	b_2	Descrição
3	0	0	1	1	Elimina a aresta formada pelos pontos b_1 e b_2
5	0	1	0	1	Se a_2 e b_2 possuem coordenadas diferentes, elimina as partes que se tocam das duas arestas relacionadas
7	0	1	1	1	Elimina a aresta formada pelos pontos b_1 e b_2 e a parte da outra aresta que toca naquela
10	1	0	1	0	Se a_1 e b_1 possuem coordenadas diferentes, elimina-se as partes que se tocam das duas arestas relacionadas
11	1	0	1	1	Elimina a aresta formada pelos pontos b_1 e b_2 e a parte da outra aresta que toca naquela
12	1	1	0	0	Elimina a aresta formada pelos pontos a_1 e a_2
13	1	1	0	1	Elimina a aresta formada pelos pontos a_1 e a_2 e a parte da outra aresta que toca naquela
14	1	1	1	0	Elimina a aresta formada pelos pontos a_1 e a_2 e a parte da outra aresta que toca naquela
15	1	1	1	1	Elimina as duas arestas

Tabela 3: Descrição de relações de eliminação de arestas. Note que há relações que, dependendo da disposição dos vértices, pertencem a mais de um tipo de relação.

Using Navigation Meshes to Improve Storytelling Dramatization

Vinicius da Costa de Azevedo

Eduardo Ceretta Dalla Favera

Cesar Tadeu Pozzer

UFSM, Departamento de Eletrônica e Computação, Brazil

Abstract

In this paper, we focus on techniques to enlarge dramatization time in an Interactive Storytelling system, allowing, at the same time, the user to give tips defining preferences to be taken into account during the dramatization. The environment where the plot is to be dramatized was expanded and divided in regions and sub-regions, being all levels represented by graphs. We adopted a weak user intervention approach during dramatization, by means of which users can specify actors or kinds of scenes they prefer to watch.

Keywords: Camera techniques, agents, storytelling, Interactive TV

Authors' contact:

{pozzer, favera, azevedo}@inf.ufsm.br

1. Introduction

Interactive Storytelling is a research area in digital entertainment that has received growing interest recently. For entertainment, the generated stories serve to guide the dramatization of events inside games, and also for replacing hand-coded pre-established scripts, with a better chance to introduce interesting and surprising variations. Another use is the application of interactive storytelling in the context of digital TV, where users would play, in principle, a more passive role than in games, but may still want to interact with the story in diverse ways.

When designing techniques for applications where the user essentially behaves as an interactive spectator, like in an interactive TV setting, we must pay special attention to both interaction and dramatization. It is expected that the story has a structure similar to conventional TV programs (in respect to duration) and, at the same time, the user should be able to interact with during dramatization.

In this paper we focus especially on alternatives of weak intervention during dramatization, as well as means to enlarge dramatization time. These alternatives have been applied in a new version of the Interactive Storytelling Logtell system, which has been developed to run in an interactive TV environment.

Section 2 presents some related work and interactive storytelling models. Section 3 gives an overview of the Logtell approach. Section 4 explains how events are now detailed in order to enhance their dramatization. Section 5 describes the specification of

the scenario and new routing strategies that have been incorporated. User interaction at the dramatization level is presented in Section 6. Section 7 describes the director architecture. Section 8 contains concluding remarks.

2. Related Work

Different approaches to digital storytelling have been proposed. The suitability of each approach depends on the goal of each application. There are two main approaches: plot- and character-based.

In a *character-based* approach [Cavazza 2002] the storyline usually results from the real-time interaction among virtual autonomous agents that usually incorporates a deliberative behavior. The main advantage of a character-based model is the ability of anytime user intervention, which means that the user may interfere with the ongoing action of any character in the story, thereby altering the plot as it unfolds. For digital TV purposes, such a strong intervention requires an excessive user interaction.

By contrast, in *plot-based* models [Ciarlini 2005], characters that usually incorporate a reactive behavior should follow rigid rules specified by a plot. The plot is usually built in a stage that comes before dramatization. In a pure *plot-based* approach, user intervention is usually more limited. Such approach ensures that actors follow a predefined script of actions which are known beforehand. This approach best fulfills our requirements for light user intervention as well for planning dramatization in advance.

3. Story Generation

Logtell [Ciarlini et al. 2005] is a system that encapsulates story generation and visualization. Story generation is based on the logic specification of a model of the chosen story genre, where possible actions and goals of the characters are described.

In the context of fairy tales that was the example used to validate the Logtell approach, possible events generated by IPG are: Reduce_Protection, Go, Get_Stronger, Kidnap, Attack, Fight, Kill, Free and Marry. Possible characters are the knights Brian and Hoel (the heroes), Princess Marian (the victim) and Draco (the villain).

The following classical story can be generated by IPG: "The protection of Marian's castle is reduced. Draco regards that as an opportunity to kidnap her. Draco then goes to Marian's Castle, attacks the castle and kidnaps Marian. As a noble knight, Brian feels compelled to save her. He goes to Draco's Castle, attacks Draco's castle twice and then fights Draco. Finally, Brian kills Draco and frees Marian, who starts loving Brian as a result. Motivated by their mutual affection, Brian and Marian go to the church and marry each other."

4. Detailing Actions for Dramatization

In the new version of the system, we adopted the overall architecture of Logtell but we decided to implement the graphical engine again and change the way a scene is represented. Events are broken into atomic actions, that provide a higher level of detail and controlled by a nondeterministic automata specified for each kind of event [Doria et al. 2008].

Atomic actions are the real actions that are performed by the actors. Walk, kiss, talk, kill, are examples of this type of action. Each one of them is performed by the actors by means of minor operations, such as movement behaviors, animations, and so on. Not all possible atomic actions specified for an event are performed. Just a few of them are really necessary to represent a specific event. These **key actions** keep the essence of the operation and cannot be omitted.

5. Scenario Architecture

One major purpose of this paper is to find means of enlarging dramatization time and enriching graphical representation. The scenario organization is our key element. A 3D representation of the modeled world was developed using the Ogre 3D engine and is compounded by various scenarios. Each one exhibits part of a fairy-tale medieval-like world.

In contrast with the original 3D scenario representation presented in Logtell, the new conception distances were strongly increased to represent real traveling distances. This approach required changes in the way scenes are filmed. Filming continuously while an actor goes from one place to another, without cuts, would result in a tedious experience, since now it might take several hours to perform an event of type "Go".

To avoid this dreary experience, it was developed a logical structure connecting scenarios: a treeph (tree + graph) (Figure 1). A treeph is a connected graph, where each node has children that form a treeph. The structure takes advantage of both data structures: the tree hierarchical organization, which provides an organized way for dealing with levels of detail; and the graph suitability in path-finding algorithms.

Each node at Level 0 represents a scenario (continuous 3D area). Alternatively, each node, at any level of the treeph structure can be denoted as a **region**. If a region offers a number of possibilities to increase actor interaction, it must be detailed and divided into **sub-regions**, which would correspond to lower levels in the tree hierarchy. For example, a desert, when compared with a city, would not require so many refinements.

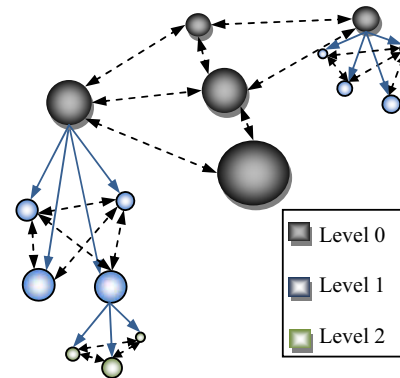


Figure 1: 3D perspective of the treeph

The regions and their sub-regions take advantage of the tree hierarchy. Children nodes inherit their parent's features since they belong to the same physical region.

5.1 Regions and Attributes

Each region has some associated attributes that characterize it and are used to select routes to represent events of type "Go", according to user's preferences, as presented in the following sections. Each attribute can be set to values within a numerical range. The lower the value, the greater is the region's level for this attribute. With this approach, path-finding algorithms can search for shortest (lightest) routes that respect user preferences.

In the current implementation there are four attributes:

- **Beauty:** How visually pleasant that region is. A waterfall or a green forest can be considered beautiful regions (lower values), in contrast with abandoned cities or garbage dumps.
- **Terror:** How creepy that region is. A dark forest should have a low value.
- **Safety:** How safe the region is. Safe places can be the princess's castle, in contrast with Draco's cave.
- **Population density:** How populated the region is.

5.2 Navigation Meshes

Each leaf node of the treeph was assigned to a set of waypoints that represent possible places actors can reach. With this approach, geographical path

information is going to be located only at the bottom of the treeph.

Besides being absolute reference position, waypoints provide useful information to the dramatization - actor's placement and camera takes - as they can provide information that different transitions between places require. Transitions between points that are physically connected, as frontiers from neighboring sub-regions, should be more subtle than transitions from different regions that are not connected. Additionally, in the last case, we need to deal with the loading time of scenarios. Wherefore, the waypoints are divided in three types.

- Path waypoint – The most common type. They are located inside leaf nodes and just provide information that guides actors to cross through the terrain.
- Short-transition waypoint – They are located in the border of the sub-regions and provide information about transitions between them. When the actor comes to a short transition waypoint, the virtual director (see Section 7) knows that is needed to perform a visual transition, which indicates that the actor is leaving one region and arriving into another. These waypoints are always pair-wise.
- Long-transition waypoint – They are located in the border between the regions of level 0. These regions require a different transition behavior with some parameters that enable the system to represent a transition between different contexts or places that are distant.

5.3 Selecting Routes

The aforementioned hierarchy of regions and attributes was designed in particular to support actions related to the movement of actors across the scene. Whenever an actor has to go from one place to another, it needs a path to be followed. We adopted an approach based on Dijkstra and A* [Buckland 2005] algorithms that finds a path with the following features: it must be connected, without cycles and the shortest in respect to the selected user attributes.

5.3.3 General Traveling Algorithm

Algorithm 1 creates the list of waypoints that connect the StartNode (SN) to the TargetNode (TG) specified by an event of type "Go". It has to take into account possible specified attributes, regions and their sub-regions and available waypoints. It presents two functionalities:

- Perform a top-down search level, starting from level 0. In each level it uses Dijkstra path-finding algorithm, searching for routes that fulfill specified attributes.

- For each leaf-node, using the A* algorithm, it finds a connected, shortest route among all the waypoints of the region. It doesn't take into account node attributes, but just real distances.

Functions `SelectStartWp()` and `SelectTargetWp()` find waypoints that are on the frontier of neighboring regions selected by Dijkstra algorithm, performed previously. `SelectStartNode()` and `SelectTargetNode()` select neighboring sub-regions that are connected by an graph edge of a subsequent lower level region.

Algorithm 1: Path-finding algorithm

```
WaypointList FindPath(SN,TN)
  Run Dijkstra(SN,TN) to find a route
    according user preferences
  For each node
    If node is a leaf
      wpS = SelectStartWp()
      wpT = SelectTargetWp()
      Run AStar(wpS, wpT)
      For each waypoint
        Add to the current path
        If waypoint is the target
          Return path
        EndIf
      EndFor
    Else
      SN = SelectStartNode()
      TN = SelectTargetNode()
      FindPath(SN, TN)
    EndIf
  EndFor
End
```

6. User Interaction

Once the route through regions can be selected in accordance with the values of their attributes, a weak intervention is granted and the generation of customizable digital content is possible. With just a few interactions, the user can change completely the dynamic of the plot representation. The user can intervene in two different (but complementary ways):

- By selecting the attributes to be considered; and
- By selecting the main actor, whose character should be emphasized.

With this approach, the same plot can be displayed in various different ways. The user could, for instance, witness the drama lived by the Hero to rescue the Princess being presented in beautiful scenarios that the protagonist would pass through. Alternatively, the malefic plans of the villain being thwarted by the Hero could be demonstrated in a terrifying world outlook.

7. The Director

In traditional film production the director is responsible for the overall decision making process. Automatic cinematography faces however two difficulties not found in the real world: the informal description of the rules of film-making are not explicit enough to be

directly encoded as a formal language. Moreover, most filmmakers work from a script that are read in advance and thus they have the opportunity to edit all the raw footage as a post process, not having to worry about user interaction like we propose here.

Our director is the agent that is responsible for the overall management of the plot dramatization, as shown in Figure 2. It communicates with the plot manager to receive the list of events that the user selected to dramatize. From the user, it obtains parameters such as preferable actor and desirable regions' attributes that should be considered to guide choices during the dramatization. The path-planner (see Section 5) is invoked to generate a set of waypoints that form a route an actor should follow, according the specified plot. Each actor receives a detailed list of atomic actions to be performed and finally, the camera module reasons about the better camera placement for each situation [Halper et al. 2001].

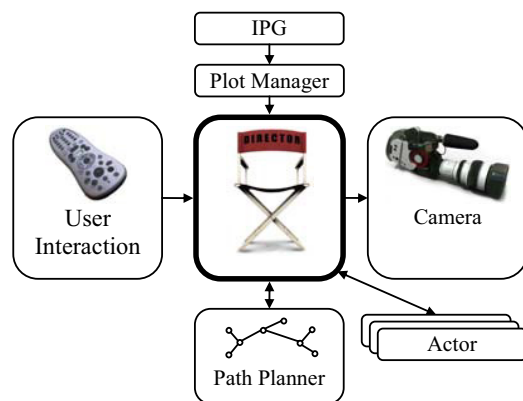


Figure 2: Director's architecture

The following algorithm summarizes the role of the virtual director proposed.

Algorithm 2: The director

```

1 Acquire the list of events from Plot Mngr
2 For each Event
3   If Event is of type "Go"
4     Invoke Path Planner to build the
       waypoints list
5     Collapse waypoints if necessary
6   EndIf
7 Define a set of atomic actions
8 For each atomic action
9   Place involved actors in the scene
10  Delegate them appropriate actions
11  Give camera parameters
12  Temporize events
13 EndFor

```

We adopt two strategies to remove unimportant and repetitive takes: we introduce the concept of Short- and Long-Transition waypoints, as presented in Section 5, and assign a level of importance to each event according to user specifications and the logic of the plot. Both are implemented in the line 5 of Alg. 2.

Moreover, the director can highlight events of type "Go" of any character in any context if the user selects this actor as the main one to be emphasized.

8. Concluding Remarks

The use of distinct regions to represent the scenario favors the diversity of the scenes that are graphically represented and provides the means to move actors across the environment extending the duration of an event according to the convenience of the dramatization.

The use of the path-finding strategy, based on weighted-graph search, also brings a good solution to highlight features according to user preferences. Our test scenario is still small, composed of just a few places. However, it was enough to test the concepts of regions with associated attributes, refinements of events of type "Go" and ways of user intervention prior and during the dramatization.

The weak user intervention proposed proved to be appropriate and effective to be applied in contexts where user is not expected to interact continuously.

9. References

- AZEVEDO, V. C., POZZER, C. T., Creating a Director for an Interactive Storytelling System. In: VI Brazilian Symposium on Computer Games and Digital Entertainment, November 2007.
- BUCKLAND, M. Programming Game AI by Example. Wordware publishing Inc, 2005.
- CAVAZZA, M., CHARLES, F. and MEAD, S., 2002. Character-based interactive storytelling. IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24.
- CIARLINI, A., VELOSO, P., and FURTADO, A., 2000. A Formal Framework for Modelling at the Behavioural Level. In Proc. The Tenth European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselkä, Finland.
- DORIA, T. R., CIARLINI, A. E. M., ANDREATTA, A. A Nondeterministic Model for Controlling the Dramatization of Interactive Stories. In: Proc. ACM MM2008 - 2nd ACM Workshop on Story Representation, Mechanism and Context - SRMC08, 2008, Vancouver, Canada, 2008.
- CIARLINI, A., POZZER, C. T., FURTADO, A. L. AND FEIJÓ, B., 2005. A logic-based tool for interactive generation and dramatization of stories, ACM SIGCHI International Conference on Advances in Computer Entertainment Technology - ACE 2005, Valencia, Spain, 133-140.
- HALPER, N., HELBING, R., STROTHOTTE, T., 2001. A camera trade-off between constraint satisfaction and frame coherence. in Eurographics, volume 20.

Construindo Cenários de Jogos Móveis Utilizando o 3DS Max e a API M3G

Bruno M. Arôxa Artur F. Mittelbach

C.E.S.A.R., Brasil

Abstract

This paper presents a 3D scene building process for mobile games. The level design built in 3D Studio Max tool is exported to a properties file which is loaded and mounted during the game – that relies on the M3G (Mobile 3D Graphics) API. Memory constraints (inherent from mobile device) were addressed providing a repository of 3D objects used to dynamically compose the game scenes. In a similar way, the “rendering window” technique was applied in order to improve game performance.

Resumo

Este artigo apresenta um processo para a construção de cenários de jogos 3D para dispositivos móveis. O *level design*, feito no 3D Studio Max, é exportado em um arquivo de propriedades que é carregado e montado sob demanda durante a execução do jogo (que utiliza a API M3G – *Mobile 3D Graphics*). As limitações de memória (características neste tipo de dispositivo) foram contornadas com a definição de um repositório de modelos utilizado para a composição dinâmica dos cenários. De maneira semelhante, a técnica de “janela de renderização” foi aplicada para otimizar o desempenho do jogo.

Keywords: construção de cenários, level design, dispositivos móveis, 3ds max, m3g

Authors' contact:

{bruno.aroxa, arturfm}@gmail.com

1. Introdução

Segundo Phill Co [2006], o *level design* é uma parte integral do desenvolvimento de um jogo. O projetista do nível precisa trabalhar junto ao time de arte para criar experiências as mais interessantes possíveis para o jogador. Além disso, ele precisa estar em contato com o *game designer* para garantir o balanceamento ideal da dificuldade do jogo, proporcionando um melhor sentimento de progresso e realização para o jogador.

Na busca por soluções para resolver os problemas característicos de jogos para dispositivos móveis (restrições de memória e baixo poder de processamento), e automatizar o processo de carga dos cenários no jogo, foram estabelecidas algumas etapas bem definidas durante o desenvolvimento do jogo. No contexto deste trabalho, esse conjunto de etapas está

sendo citado como “processo”. Como estudo de caso, o processo proposto foi aplicado na construção dos cenários (e *level design*) de um jogo 3D de naves, rodando em um dispositivo sobre a plataforma Java (CLDC [2008] 1.1, MIDP [2008] 2.0 com suporte à API M3G [2008]).

Este artigo está estruturado da seguinte forma: a seção 2 cita outros trabalhos relacionados com o tema proposto; a seção 3 apresenta as ferramentas envolvidas na construção dos cenários; a seção 4 detalha as etapas do processo de construção dos cenários; finalmente, a seção 5 expõe as conclusões e considerações finais deste trabalho.

2. Trabalhos Relacionados

O projeto de motores para exportar cenários do 3DS Max para jogos 3D não é uma idéia nova. Watsa [2001] propôs a utilização do Max Script para gerar níveis para jogos dentro do próprio 3DS Max. Para o contexto do estudo de caso feito por Watsa, a proposta foi de modificar o 3DS Max, através de *plugins* e *scripts*, para automatizar tarefas de criação de níveis.

Outro trabalho mais recente [Salvi et al. 2006] propõe a utilização do 3DS Max como editor de níveis para jogos 3D utilizando o motor de processamento gráfico OGRE [2008]. O trabalho utilizou como estudo de caso o jogo para PC “Taltun: A terra do conhecimento”.

Apesar da proposta de utilizar o 3DS Max como editor de nível ser similar aos trabalhos citados, este trabalho se difere tanto no aspecto de exportação dos níveis como no processo de carga dos cenários, que precisa satisfazer exigências da API 3D para dispositivos móveis.

3. Ferramentas Utilizadas

O desenvolvimento de jogos requer uma gama variada de ferramentas que vão desde ferramentas de controle de versão (CVS – *Concurrent Version System*, SVN – *Subversion*, etc.) até ferramentas de modelagem e edição de imagem. Esta seção abordará apenas as ferramentas envolvidas na construção de cenários e no *level design*.

3.1. 3DS Max

O 3DS Max [2008] foi escolhido como ferramenta para criação dos níveis por cinco motivos básicos:

- O projeto já possuía licença do software;
- O time já tinha know-how da ferramenta;
- O time já tinha know-how dos exportadores M3G [2008];
- Vasta biblioteca de material de pesquisa;
- O suporte acessível à criação de scripts de exportação.

3.2. Max Script

O Max Script é a ferramenta de *scripts* própria do 3DS Max. Através dela é possível criar, ler e redefinir atributos para os objetos da cena, bem como exportá-los para arquivos de propriedades. Além disso, a ferramenta também permite a criação de interfaces gráficas para facilitar a execução dessas tarefas.

O Max Script foi escolhido pela sua vasta biblioteca de referência [Max Script 2008], integração total com o 3DS Max e flexibilidade.

3.3. M3G Exporter

Para o processo de exportação do repositório de peças, feito com base no formato M3G, foi utilizado o pacote de ferramentas de desenvolvimento disponibilizado pela Mascot Capsule [2008]. O pacote inclui:

- *Micro3D Plugin*: para exportação através do 3DS Max;
- *M3G Converter*: Ferramenta para a conversão do arquivo exportado pelo *Micro3D Plugin* (.h3t) para o formato carregado pela API M3G no jogo;
- *M3G Viewer*: Ferramenta de visualização de arquivos no formato m3g.

Apesar de algumas limitações, a ferramenta foi escolhida por falta de uma ferramenta alternativa que se aproximasse das necessidades do projeto

4. Processo para construção dos cenários

O processo de criação de cenários adotado neste trabalho apresenta seis etapas bem definidas que são ilustradas na Figura 1.

Na primeira etapa são criadas as peças que servirão de base para o *level design*. Com o repositório estabelecido, segue-se a segunda etapa que diz respeito à construção do cenário propriamente dito. As etapas seguintes envolvem as exportações dos cenários e do repositório, preparando os artefatos necessários para a carga do cenário no jogo. Ao final do processo, o *level design* é validado com o intuito de proporcionar a melhor experiência possível para o usuário.

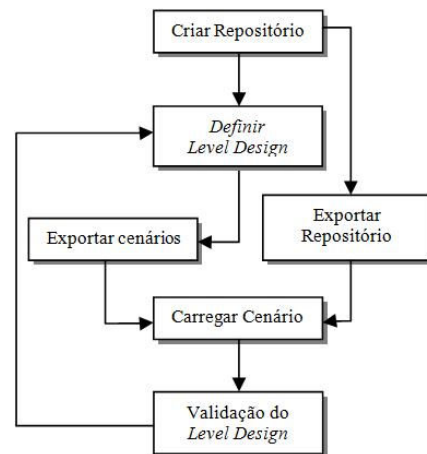


Figura 1: Etapas do processo de criação de cenários

4.1. Criar Repositório

Uma das preocupações nessa etapa é de projetar peças que possam ser usadas em situações diferentes, sem que se perceba a repetição, fato que é possível com as manipulações básicas – translação, rotação e escalonamento [Clua e Bittencourt 2005] – disponíveis no 3DS Max.

Para construção de estruturas inorgânicas como os prédios do jogo foram usados sólidos básicos, tais como: cubos, paralelepípedos, etc. Para a criação de figuras orgânicas de cenário como montanhas, arbustos, corais, etc., foi modelada uma peça irregular simples, porém, que permitisse a sua utilização em diferentes contextos. Para facilitar futuras referências, essa peça metamórfica, cuja estrutura básica é apresentada na Figura 2, será chamada de *brush*.

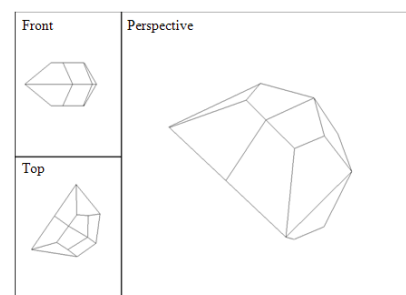


Figura 2: Sólido geométrico, base para composição dos cenários

4.2. Definir Level Design

O processo de concepção do *level design*, no contexto deste jogo, seguiu o seguinte fluxo:

1. *Definir o tema do nível*: A definição do tema dá suporte para a arte conceitual do cenário;
2. *Definir o skyline do nível*: Diz respeito às características do cenário de acordo com o tema definido previamente;
3. *Montar a estrutura básica do cenário*: Definir a sequência de terrenos, utilizando as peças do

repositório, com base na duração estimada do nível;

4. *Montar elementos orgânicos*: Definir os elementos estáticos do cenário utilizando modificações do *brush*;
5. *Inserir marcadores para inimigos*: Definir cada inimigo bem como seus atributos (velocidade, inteligência, formação, etc.);
6. *Inserir marcadores para itens coletáveis*: Posicionar os itens coletáveis de acordo com a geografia e dificuldade do nível.

4.3. Exportar Cenário

O processo de exportação do cenário consiste unicamente na extração de propriedades relevantes dos objetos no cenário, de forma a possibilitar a sua reconstrução no ambiente do jogo. Para diminuir a complexidade do problema, o cenário foi dividido em unidades lógicas, chamadas *terrenos*. Os terrenos, por sua vez podiam conter ou não objetos 3D. Para cada objeto foram atribuídas as seguintes propriedades:

- Indexador que o associa ao terreno;
- Identificador do tipo de objeto;
- Coordenadas de posição;
- Valores de rotação;
- Informação de escala;
- Identificador de textura.

Além destas propriedades, padrão para todos os objetos, os objetos dinâmicos (inimigos, itens coletáveis, etc.) acumulavam ainda um conjunto de atributos específicos (velocidade, tipo, etc.).

A extração das informações dos objetos para um arquivo de propriedades foi feita utilizando o Max Script – ferramenta de *scripts* que integra o próprio 3DS Max., o respeitando um protocolo simples, especificado pela própria equipe para definir a ordem de escrita das propriedades. A definição desse protocolo foi fundamental para garantir a leitura correta dos arquivos de propriedades na etapa de carga do cenário, durante o jogo. A Figura 3 apresenta um exemplo de nível exportado.

```
23,05,1.0,0.5,1.0,133.445,0.0,-1.0,0.0,-30.5798,54.576,11607.1,1
23,05,1.0,1.37294,1.0,60.8267,0.0,-1.0,0.0,-120.228,-20.6941,11620.6,1
23,05,1.0,1.97294,1.0,55.8509,0.0,1.0,0.0,70.6496,-6.15476,11677.3,1
23,05,1.78331,1.0,1.0,59.7451,0.0,-1.0,0.0,-96.4743,1.58681,11733.5,-1
23,05,1.78331,1.0,1.0,59.7451,0.0,-1.0,0.0,-85.3158,1.58681,11740.0,-1
23,800,1.0,1.0,1.0,0.0,0.0,0.0,0.0,15.4688,44.2274,11783.3,0
```

Figura 3: Entradas de um arquivo de nível exportado

A entrada em destaque diz que será adicionado no terreno 23 (primeira propriedade da linha) que um *brush*, cujo tipo está indicado pela segunda propriedade, com as características de escala, rotação e posição especificadas nas propriedades seguintes. A última propriedade define a textura que será aplicada ao *brush*.

4.4. Exportar Repositório

O repositório exportado (arquivo *.m3g*) contém um conjunto de elementos que podem ser carregados dinamicamente no jogo através da API 3D. Para a visualização desses arquivos foi utilizada a ferramenta M3G Viewer, que faz parte do pacote de ferramentas disponibilizado pela Mascot. A Figura 4 apresenta o esquema de visualização da estrutura da árvore de nodos gerada pelo exportador.

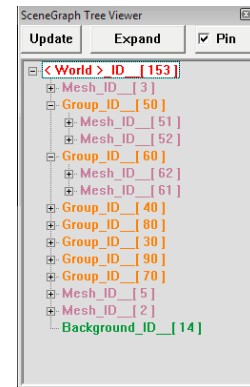


Figura 4: Interface gráfica do exportador

Um dos maiores benefícios trazidos por este visualizador é a possibilidade da validação dos objetos antes mesmo da carga do arquivo na aplicação. Desta forma, os problemas no processo de exportação eram identificados com antecedência, tornando o trabalho mais produtivo.

4.5. Carregar o Cenário

Durante a inicialização de cada fase do jogo, o cenário é carregado com base nas propriedades dos terrenos, exportadas nas etapas anteriores. Para otimizar o desempenho do jogo e o uso de memória, a carga dos objetos é feita utilizando uma janela de renderização, que garante um limite máximo de objetos 3D carregados na cena. A Figura 5 apresenta um diagrama esquemático da janela de renderização, que para o contexto do jogo, com os devidos ajustes de câmera (*near clip*, *far clip*, *FOV – Field of View*, etc.), apresentou resultados satisfatórios com apenas três terrenos.

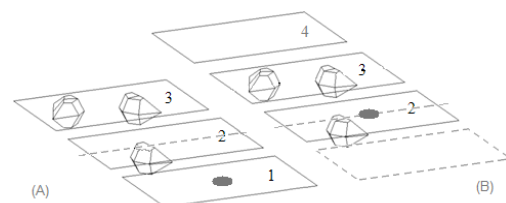


Figura 5: Diagrama esquemático da janela de renderização com três terrenos

A progressão do jogo sobre o cenário é medida pela projeção da câmera sobre o terreno (representado pelo ponto escuro na Figura 5 (A)). Quando a projeção alcança a metade do terreno seguinte (2), os objetos do terreno anterior (1) são liberados para o próximo

terreno não carregado (4) seja alocado, como é apresentado na Figura 5 (B).

Durante o processo de carga de cada terreno, o motor recupera a peça no repositório de acordo com o seu tipo e aplica as transformações necessárias, com base no arquivo de propriedades do nível. Para os objetos dinâmicos (inimigos, itens coletáveis, etc.) são analisados também propriedades específicas como: IA, velocidade, estratégia, etc. Como boa parte do tratamento dos dados é feita durante a exportação dos cenários, o módulo de carga do cenário no jogo acabou sendo bastante enxuto, com apenas duas classes para controle e entidades e um *parser* para fazer a tradução dos níveis exportados.

4.6. Validar Level Design

A validação do *level design* é feita levando-se em consideração alguns dos aspectos de jogabilidade definidos por Phill Co [2006]:

- Desafios justos (jogos muito fáceis ou muito difíceis acabam por frustrar o jogador);
- Sistema de recompensa que motive o usuário a querer jogar;
- Objetivos dentro do alcance do jogador.

Apesar de saber que idealmente um bom *level design* deve apresentar as características citadas, na prática o resultado da avaliação acaba sofrendo influência de fatores externos como: TTM – *Time to Market* e limitações dos aparelhos (viabilidade técnica).

5. Conclusão

A utilização do processo para a construção de cenários, proposto neste artigo, se mostrou bastante satisfatória e eficaz durante o desenvolvimento do jogo. Os resultados obtidos apresentaram um ganho expressivo no tamanho final do jogo. O repositório exportado consumiu em torno de 30kb e aproximadamente 15kb (valor médio) por arquivo de propriedade (*level design*). Em testes realizados com a exportação do nível completo, um único cenário ficou com aproximadamente 115kb, fato que inviabilizou esta abordagem (visto que foram construídos nove cenários).

Além do ganho no tamanho do jogo, o uso do motor proporcionou um melhor uso dos recursos de memória do aparelho. Com a abordagem de janela de renderização, foi possível ter um controle mais fino da quantidade de objetos 3D carregados na memória.

Apesar dos pontos positivos mencionados, alguns desafios precisam ser melhorados, tais como: melhorar a GUI do exportador, permitindo a inserção e edição das propriedades dos objetos; reduzir a precisão dos valores das propriedades exportadas, que atualmente

estão com cinco casas decimais, para apenas duas (que já seriam mais que suficientes para o jogo desenvolvido), de forma a reduzir o tamanho dos arquivos de nível.

Agradecimentos

Os autores agradecem ao C.E.S.A.R [2008] pela excelente infra-estrutura provida para a execução e sucesso deste projeto.

Referências

- 3DS MAX, 2008. Disponível em: <http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112> [Acessado em 14 Ago. 2008].
- MAX SCRIPT, 2008. Disponível em: http://images.autodesk.com/adsk/files/maxscript_8_help.zip [Acessado em 14 Ago. 2008].
- C.E.S.A.R, 2008. Disponível em: <http://www.cesar.org.br> [Acessado em 12 Ago. 2008].
- CLDC – CONNECTED LIMITED DEVICE CONFIGURATION API, 2008. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html> [Acessado em: 03 Out. 2008].
- CLUA, E. E BITTENCOURT, J. R., 2005. Desenvolvimento de Jogos 3D: Concepção, Design e Programação. In: *XXIV Jornada de Atualização em Informática (JAI) Part of XXIV Congresso da Sociedade Brasileira de Computação*, 22-29 Jul. 2005. São Leopoldo. São Leopoldo: UNISINOS, 1313-1357.
- Co, P., 2006. *Level Design for Games: creating compelling game experiences*. Berkley, CA: New Riders, 2006.
- M3G API, 2008. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr184/index.html> [Acessado em: 14 Ago. 2008].
- MASCOT CAPSULE M3G TOOLKIT, 2008. Disponível em: <http://www.mascotcapsule.com/toolkit/m3g/en/index.php> [Acessado em: 14 Ago. 2008].
- MIDP – MOBILE INFORMATION DEVICE PROFILE API, 2008. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html> [Acessado em: 03 Out. 2008].
- OGRE, 2008. Disponível em: <http://www.ogre3d.org/> [Acessado em 18 Ago. 2006].
- SALVI, J. L., SANTOS, M. C., TORTELLI, D. M. E BRANCHER, J. D. 2006. Utilizando o 3D Studio Max como Level Editor para Construção de Cenários para Ogre3D. In: *V Workshop Brasileiro de Jogos e Entretenimento Digital parte do SBGames 2006*, 8-10 Nov. Recife.
- WATSA, S. 2001. *Case Study: Using Max Script for Building Game Levels* [online] Gamasutra – The Art & Business of Making Games. Disponível em: http://www.gamasutra.com/features/20010824/watsa_pfv.htm [Acessado em 18 Ago. 2008].

Estudo e Implementação de Heurísticas para Otimizar os Algoritmos Aplicados a *Puzzle Games*

Alex F. V. Machado Esteban W. G. Clua

Carla R. B. Bonin* Gustavo M. Novaes* Mauro L. R. de A. Filho*

Universidade Federal Fluminense, Dep. Ciência da Computação, RJ, Brasil

*Centro Federal de Educação Tecnológica, Dep. Informática Industrial, MG, Brasil

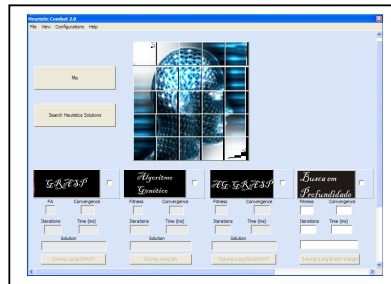


Figura 1 – Interface do *Kombat Heurístico* 2.0.

Programa/jogo de avaliação do desempenho entre heurísticas no domínio de jogos de raciocínio.

Resumo

Este trabalho propõe definir o estado-da-arte no que diz respeito a procedimentos que busquem a solução em jogos eletrônicos de raciocínio (*puzzle games*) com apenas um usuário, nos quais englobam os jogos que usam árvores de busca como modelagem do sistema, tais como os quebra-cabeças e *sudoku*, além de toda gama de jogos que utilizam, mesmo que parcialmente, *path finding*.

Para tanto, é feita uma análise comparativa das heurísticas GRASP, Algoritmo Genético e AG-GRASP, além da técnica tradicional de busca em profundidade, com base em experimentos sobre um jogo de quebra-cabeças de dimensões 5x5.

Palavras-Chave: Otimização de algoritmos, GRASP, algoritmo genético, jogos de raciocínio.

1. Introdução

Este trabalho busca a exploração das heurísticas GRASP (Generic Search Algorithm for the Satisfiability Problem), Algoritmo Genético e AG-GRASP (junção dos conceitos do Algoritmo Genético com o GRASP), em um segmento do domínio de jogos que utiliza predominantemente busca em profundidade e busca exaustiva.

A hipótese tida como base é comprovada através do *software Kombat Heurístico* (Figura 1), que foi desenvolvido durante o projeto. A realização de uma melhor análise da situação/problema existente torna-se possível através de sua interface, permitindo a alteração de parâmetros dos algoritmos para as

análises de convergência e velocidade de resolução através de testes automatizados.

Pretende-se adaptar procedimentos heurísticos a árvore de busca para a otimização de algoritmos aplicados a *puzzle games*. Dessa forma, a implementação de tais heurísticas possibilita a existência de resolvedores completos ou simultâneos (*hints*), que sejam muito mais eficientes do que os registrados na literatura. Outra aplicação indireta a ser observada é o seu uso em jogos que fazem uso da estratégia conhecida como *pathfinding*, que visa traçar a melhor rota para atingir um determinado objetivo [Pozzer 2008].

2. Trabalhos Relacionados

Em [Hong et al. 2001] é feito uma abordagem de algoritmo genético para solucionar os movimentos em um quebra-cabeças que demonstrou que tal algoritmo possui um desempenho melhor que o algoritmo determinístico de busca em profundidade de *Dijkstra*. Neste trabalho foi reconstruído e melhorado o algoritmo desenvolvido em [Hong et al. 2001], e proposta três novas abordagens ao mesmo problema: utilizando o motor GRASP puro, AG com GRASP (AG-GRASP) e busca profundidade para fins de comparação.

3. O Jogo de Quebra-Cabeças Abordado

O jogo usado como estudo de caso é uma versão do tradicional quebra-cabeça de tabuleiro, com pequenas modificações para aumentar a jogabilidade no

computador. Embora a implementação desenvolvida seja altamente escalável, foi criado um ambiente 5x5 composto de 5 peças no eixo vertical e cinco peças no eixo horizontal, totalizando 25 peças (como mostra a estrutura ordenada da Figura 2).



Figura 2 – Estrutura ordenada do quebra-cabeça

O procedimento para jogar o quebra-cabeça proposto é:

- Definir a imagem base do jogo
- Embaralhar as peças para gerar o estado inicial ou o problema
- Alterar os estados do jogo utilizando movimentos regidos por regras até a solução (posições iniciais das peças) ser alcançada

4. Solução Através de Algoritmo Genético

Tomando como base o trabalho de [Hong et al. 2001] é possível aplicar algoritmos genéticos para definir os movimentos do jogo de quebra-cabeças. A estrutura de um cromossomo pode ser representada pelo conjunto dos rótulos das associações de uma *Game Search Tree* para chegar a determinado nodo. Por meio dos cromossomos são então executadas as operações de *crossover* para gerar um grupo de *offspring* foi definida com um ponto fixo para a divisão, e de mutação para trocar elementos (genes) randomicamente de um cromossomo, aumentando a diversificação e tentando sair de ótimos locais. Outra operação realizada é o cálculo da função de *fitness*, que representa o somatório da distância, em movimentos, de cada peça até sua posição inicial.

5. Solução Através de GRASP

A base da solução do problema a partir da heurística GRASP utiliza a GST e a lista de movimentos definidos no AG. A estrutura GRASP baseia-se em:

- Função de Avaliação (FA): a mesma utilizada pela função de *fitness* no algoritmo genético. Essa foi definida como sendo também de maximização.
- Lista de Candidatos (LC): todos os movimentos possíveis. O tamanho da LC para esse problema será sempre de 50.

- Lista Restrita de Candidatos (LRC): Uma lista de E elementos contidas em LC com as melhores funções de avaliação.

As etapas do GRASP modelado a esse problema são:

- Etapa 1 - Sortear um movimento inicial (entre os 50 movimentos possíveis).
- Etapa 2 - Definir a LC.
- Etapa 3 - Calcular a FA de cada movimento (nodo) baseado no estado de alteração gerado pelo último movimento.
- Etapa 4 - Definir a LRC.
- Etapa 5 - Sortear um movimento da LRC.
- Etapa 6 - Repetir a etapa 2 até encontrar a FA máxima ou alcançar o número de movimentos máximos.
- Etapa 7 - Se a solução não for encontrada, repetir a etapa 1 em busca de uma nova outra solução mais R vezes.
- Etapa 8 - Ao final, se a solução não for encontrada, comparar a lista de soluções e definir a melhor.

Embora seja altamente recomendado o uso de um método de busca local para obter a garantia de alcançar ótimos locais de boa qualidade [Zanetti 2005], foi eliminado esta etapa para poder comparar o verdadeiro motor GRASP com as demais heurísticas.

6. Solução Através de Algoritmo Genético com GRASP (AG-GRASP)

O Algoritmo Genético possui um início aleatório no qual os cromossomos são gerados por meio de uma função randômica, sem que haja qualquer critério de refinamento, e possui um método de busca que faz operações buscando otimizar o algoritmo genético, enquanto que o método de construção do GRASP é totalmente elitista tendo a preocupação de sempre gerar soluções de boa qualidade que façam com que o algoritmo inicie de um ótimo local, e não possui um processo de busca associada ao da geração inicial.

Dessa forma, a modelagem AG-GRASP proposta para este projeto tem por objetivo utilizar o GRASP no processo de geração inicial e aplicar o Algoritmo Genético para realizar a busca local. No Algoritmo Genético proposto em [Hong et al. 2001] 10 soluções iniciais foram geradas randomicamente, em contraposição neste algoritmo é utilizado o GRASP para gerar 10 soluções iniciais.

Assim, a modelagem do AG-GRASP aplicado à *Game Search Tree* ficaria da seguinte maneira:

- Etapa 1 - Representação de todas as situações.

- Etapa 2 - Definição do tempo limite e do número de gerações.
- Etapa 3 - Definição da profundidade (game tree) e da função de *fitness*.
- Etapa 4 - Definição da taxa de *crossover* e mutação.
- Etapa 5 - Geração da população inicial de cromossomos utilizando GRASP.
- Etapa 6 - Execução do *crossover*.
- Etapa 7 - Execução da mutação.
- Etapa 8 - Cálculo do valor de *fitness* de cada offspring.
- Etapa 9 - Seleção dos melhores candidatos.
- Etapa 10 - Finalização ou repetição de todo o processo a partir da Etapa 6.

7. Solução Através de Busca em Profundidade (DFS)

A solução através de busca em profundidade foi implementada, baseando-se em explorar todas as possíveis soluções para o jogo e tem como princípio o de explorar cada nodo até que este atinja sua profundidade máxima. Desse modo, o algoritmo desenvolvido consiste em explorar todos os nodos filhos até que atinja um nodo filho que não possua outro nodo filho, também conhecido como nodo folha. Caso não seja encontrada a solução para o problema existente o algoritmo voltará no nodo filho anterior e recomeçará a executar a busca.

8. Implementação e Desenvolvimento do *Software Kombat Heurístico 2.0*

Para a implementação do software foi escolhida a linguagem *FreePascal* na ferramenta RAD (*Rapid Application Development*) *Lazarus*, que foi escolhida devido à sua característica de *cross-compiling*, a facilidade de desenvolvimento, a boa velocidade de resolução, a gratuidade e a disponibilidade da licença de forma a não limitar a expansão dos conceitos a serem observados no projeto [Monteiro et al. 2007] [Piske e Seidel 2007].

Foi então desenvolvido um sistema, o *Kombat Heurístico*, que possibilita o usuário jogar o quebra-cabeça, fazer configurações sobre os parâmetros das heurísticas ou comparar seus desempenhos. O principal objetivo desse programa é comparar o número de iterações, convergência e tempo de processamento entre as aplicações das heurísticas propostas.

8.1 Testes Realizados

Os testes realizados tiveram o objetivo de avaliar o desempenho com relação à convergência e a velocidade de resolução dos algoritmos implementados no *software* desenvolvido. Dois

parâmetros comuns às três heurísticas que merecem destaques são:

- Número máximo de iterações: Representa, no caso do algoritmo genético, o número de gerações e no GRASP o número de soluções geradas.
- Número de movimentos máximos para a heurística solucionar o problema: representa o tamanho da árvore de busca (GST).

Foram documentados os resultados para 20 jogos (ou problemas/situações).

8.2 Análise da Convergência

A convergência foi analisada utilizando o critério de percentual de situações em que foi achada uma solução (Figura 3). Um algoritmo perfeito deveria aumentar seu percentual em proporção ao aumento do número de movimentos possíveis.

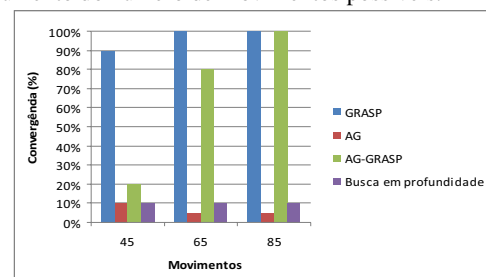


Figura 3 - Gráfico comparativo entre os percentuais de convergência das quatro heurísticas

No que diz respeito ao percentual de convergência, observou-se que a Busca em Profundidade e o Algoritmo Genético solucionaram apenas 10% dos problemas apresentados fazendo uso de 45 movimentos, enquanto que o AG-GRASP solucionou 20% dos casos e o GRASP começou a se destacar solucionando 90% dos problemas gerados. Já com 65 movimentos o motor GRASP puro encontrou uma solução em todos os casos, o AG-GRASP encontrou a solução em 80% dos casos, e o AG solucionou em apenas 5% dos testes realizados, enquanto que a Busca em Profundidade continuou resolvendo o problema em 10% dos casos. Na mais folgada das situações (85 movimentos), o GRASP se manteve resolvendo todos os casos, o AG-GRASP melhorou sua convergência resolvendo também todos os problemas e o AG e a Busca em Profundidade mantiveram seu fraco e insatisfatório desempenho.

8.3 Análise da Velocidade de Resolução

A velocidade foi analisada tendo como base o critério de tempo médio gasto (dos casos que o algoritmo conseguiu convergir) para encontrar a solução para o problema proposto. Onde o algoritmo perfeito deveria diminuir seu tempo em proporção ao aumento de possibilidades de movimento. Na Figura 4 tem-se um gráfico que demonstra o tempo gasto em

segundos para encontrar a solução para os problemas propostos.

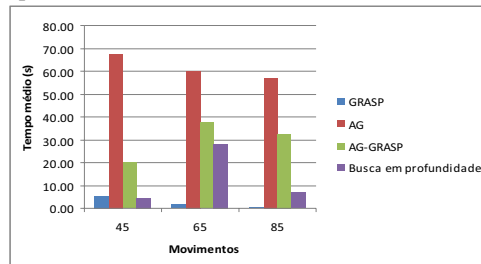


Figura 4 – Gráfico comparativo entre os tempos de convergência das quatro heurísticas

Com relação ao tempo para a convergência, o AG demanda, em alguns casos, mais de um minuto para a resolução dos problemas. A Busca em Profundidade por sua vez gasta aproximadamente 5 segundos para resolver com 45 movimentos, 28 para resolver com 65 e 7 para resolver com 85 movimentos.

Através dos gráficos é possível definir que o GRASP possui também uma maior velocidade pois demanda com 45 movimentos aproximadamente 5 segundos, com 65 movimentos apenas um segundo e em problemas que fazem uso de 85 movimentos para resolução é capaz de resolver em um tempo inferior a um segundo. Já o AG-GRASP, gasta um tempo muito superior devido à suas características herdadas do AG, o que a torna uma heurística imprecisa para os padrões desejados por esta pesquisa.

8.4 Análise do Desempenho

Dessa forma, tem-se que a heurística GRASP foi a que se apresentou melhor adaptada a tal problema proposto, sendo assim, é possível afirmar que esta possui o melhor desempenho aplicada ao domínio de jogos tratado neste estudo. Considerando-se ainda haver uma proporção direta entre o tempo gasto e o número de iterações executadas, deduz-se que o GRASP ainda teria o menor gasto de memória.

9. Conclusão

Neste trabalho foram desenvolvidos dois novos algoritmos para busca em árvores no domínio de jogos, mais especificadamente quebra-cabeças. O primeiro é baseado na heurística GRASP pura, apoiado principalmente em suas características randômicas e gulosas de busca. O segundo foi desenvolvido usando o princípio construtivista do GRASP e o algoritmo genético como busca local. O AG e a busca em profundidades foram também implementados para dar credibilidade a nossa tese.

Foi criado um *software* para testar as quatro rotinas, denominado *Kombat Heurístico*, o qual serviu como base para a geração das tabelas

comparativas e gráficos, para a avaliação da convergência e velocidade de resolução.

Portanto, após a revisão bibliográfica efetuada, o desenvolvimento do projeto através da implementação do *software Kombat Heurístico*, a realização de testes e a análise dos resultados obtidos foi possível concluir que o GRASP é a heurística que melhor se adaptou ao problema proposto.

O procedimento GRASP sugere duas contribuições de grande relevância para a área de computação em jogos:

- Inovação através de um algoritmo inteligente em uma área que usa predominantemente força bruta.
- Nova oportunidade para a criação de jogos mais confiáveis (por conseguirem analisar a possibilidade de achar a solução), com menos uso de memória e maior velocidade de processamento, portanto tornando possível o desenvolvimento de jogos com melhor utilização do hardware.

Referências

- HONG, P. T. ET AL., 2001. Applying Genetic Algorithms to Game Search Trees. Soft Computing.
- MONTEIRO, FELIPE; BIASA, CESAR; FÉLIX, JOSÉ, 2007. Osciloscópio Digital em Placa ISA. 2007
- PISKE, O. R.; SEIDEL, F. A., 2006. Rapid Application Development, 2007.
- POZZER, C. T., 2008. Teoria dos Grafos. Universidade Federal de Santa Maria, 2008.
- ZANETTI, MÁRCIA CRISTINA VALLE, 2005. Ochi, Luiz Satoru. Desenvolvimento e Análise Experimental da Heurística Grasp Aplicada a um Problema de Coleta Seletiva. XXXVII SBPO 2005, Gramado/RS.

Estratégias de Interação para Storytelling na Televisão Digital

Marcelo M. Camanho Angelo E. M. Ciarlini
Antonio L. Furtado* Cesar T. Pozzer[‡] Bruno Feijó*

UNIRIO, Dep. de Informática Aplicada, Brasil *Puc-Rio, Dep. de Informática, Brasil

[‡]UFSM, Dep. de Eletrônica e Computação

Resumo

O Storytelling Interativo, técnica que permite a geração, visualização e controle de histórias interativas, é uma forma de entretenimento que parece promissora para uso na TV digital. Um dos principais obstáculos para o uso em tal ambiente é achar um meio de gerar uma história conforme os usuários a assistem, e permitir que eles possam interferir com o que está acontecendo, sempre mantendo a história com um certo nível de coerência e responsividade. Este artigo descreve um novo modelo de Storytelling Interativo para lidar com os requisitos necessários de interatividade dinâmica em um ambiente de televisão digital. O modelo é baseado na extensão do Logtell, um sistema já existente, que utiliza lógica formal para a geração e dramatização interativa de histórias.

Keywords: Interactive Storytelling, Artificial Intelligence, Automated Planning, Digital Television, Logics.

Authors' contact:

marcelo.camanho@uniriotec.br
angelo.ciarlini@uniriotec.br
*furtado@inf.puc-rio.br
‡pozzet@inf.ufsm.br
*bfeijo@inf.puc-rio.br

1. Introdução

Um sistema de storytelling interativo em mídias como a Televisão Digital, além da dificuldade de equilibrar coerência e interatividade, precisa de alta responsividade na hora da geração e controle das histórias, e deve considerar o fato de que o usuário pode nem mesmo querer interagir de modo algum. O sistema precisa ter tanta responsividade quanto possível e a história deve ser fluente, sempre mantendo a simplicidade e a sensação de conforto na interação, especialmente considerando que as histórias em geral serão assistidas por espectadores que podem não ser ávidos usuários de jogos eletrônicos. Ao usuário deve ser permitido interagir de acordo com seu gosto, mas o sistema de storytelling deve manter a coerência como um jogo convencional, pois o objetivo é assistir boas histórias onde se pode interferir, e não vencer um desafio.

O Logtell[Ciarlini et al. 2005] é um sistema de storytelling interativo cuja abordagem parece ser boa para TV, pois se centra em manter as histórias

coerentes enquanto permite que o usuário possa interferir na história em diferentes níveis. O usuário pode escolher não interferir na história ou ter uma participação mais forte, estabelecendo, por exemplo, que eventos ou situações ocorram, desde que sejam logicamente coerentes com o modelo especificado para o gênero de história que está sendo usado. Uma das características do Logtell atual que não é apropriada para a televisão digital é que, nele, a geração e a dramatização da história são separadas, concentrando-se a interatividade na fase de geração. Para levar o Logtell para um ambiente de TV interativa, é preciso realizar essas atividades em paralelo, e também encontrar mecanismos confortáveis para a interação.

Alguns sistemas de storytelling interativo se focam na interação de personagens autônomos (character-based) [Cavazza et al. 2002], o que facilita a interação, outros focam-se na trama, com regras mais rígidas que garantem a coerência [Cavazza et al. 2002; Charles 2001]. Outros como [Mateas and Stern 2000] tentam acomodar ambas as características. O uso de técnicas de planejamento automatizado, seja com redes hierárquicas de tarefas (HTN) [Cavazza et al. 2002] ou com algoritmos mais flexíveis [Riedl and Young 2004] é parte importante de alguns sistemas de storytelling interativo, criando seqüências lógicas de eventos. No trabalho em questão usa-se uma abordagem que combina enfoques e técnicas de modo a se obter boa diversidade de histórias com um nível de interação adequado para TV Digital Interativa.

2. Abordagem do Logtell

A abordagem do Logtell procura gerar e dramatizar histórias variadas de um determinado gênero, tomando como base uma especificação em lógica formal. A idéia é permitir que o usuário possa interferir à vontade na história desde que se mantenha o sentido e a coerência. A trama é gerada por um módulo chamado IPG (Interactive Plot Generator). Os enredos são gerados em múltiplos estágios que alternam fases de inferência de objetivos, planejamento e interação com o usuário. O Logtell concilia aspectos *plot-based* e *character-based*. O contexto para a geração de histórias é acessado através do Context Control Module (CCM), e contém a descrição lógica formal do gênero das histórias a serem geradas e seu estado inicial. O gênero é descrito por um conjunto de operações parametrizadas com pré- e pós-condições, especificando quais eventos podem ocorrer; e um conjunto de regras de inferências de objetivos

especificadas com uma lógica temporal modal, declarando situações que levam os personagens a buscarem objetivos. Para garantir a coerência, a interação é sempre indireta. O usuário pode deixar a história continuar ou pedir alternativas. Os modos de interação são explicados em maior detalhe na seção 3. A dramatização da história não ocorre em tempo real em relação à geração da trama, podendo ser ativada após parte da trama já ter sido criada, ou quando já estiver completa. Esta dramatização é feita com atores 3D, usando um motor gráfico próprio que foi criado para garantir a consistência do modelo lógico das tramas com a dramatização.

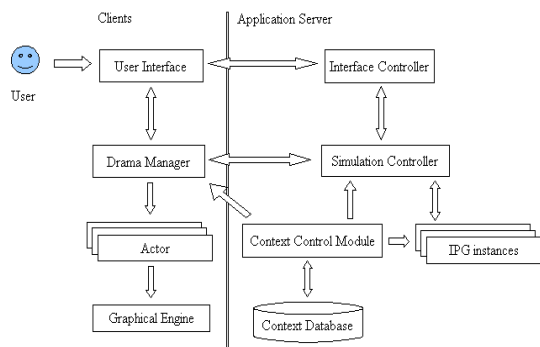


Figura 1. Nova Arquitetura do LOGTELL.

Na nova arquitetura, apresentada na Fig. 1, o Logtell é modificado para um modelo cliente-servidor. O lado do cliente agora é responsável pela interação e dramatização das histórias. O lado do servidor é responsável pela simulação. Os processos agora ocorrem em paralelo e devem ser coordenados entre si. Para cada história que está sendo apresentada, há uma aplicação em execução no servidor. Na TV interativa digital, os *set-top boxes* certamente terão recursos computacionais limitados. Ao concentrar a simulação em servidores de aplicações, têm-se uma melhor escalabilidade. Além disso, é mais fácil exercer controle quando uma única história é compartilhada por vários usuários.

Na versão anterior do Logtell, a interface principal do sistema concentrava o controle geral da história, utilizando e ativando uma instância do IPG e acionando o Drama Manager. Na nova arquitetura, esta interface se decompõe em 3 módulos de controle: o Simulation Controller, o Interface Controller e a User Interface. O Simulation Controller e o Interface Controller são executados no servidor e a Graphical User Interface (interface do usuário) no cliente. No lado do cliente, o usuário interage com o sistema através da User Interface, que informa as interações desejadas ao Interface Controller localizado no servidor. O Drama Manager requisita os eventos a serem dramatizados ao Simulation Controller, e controla os atores 3D que representam cada personagem na nossa implementação gráfica. No lado do servidor, o Interface Controller centraliza sugestões às histórias fornecidas por vários clientes. Quando múltiplos usuários compartilham a mesma história, as interações mais votadas são tentativamente inseridas.

Quando há somente um cliente, as sugestões são automaticamente enviadas ao Simulation Controller. O Simulation Controller é responsável por: informar ao Drama Manager no lado do cliente os próximos eventos a serem dramatizados; receber pedidos de interação e incorporá-los à história; selecionar interações viáveis e possivelmente interessantes, para serem disponibilizadas aos usuários; controlar um número de instâncias do Interactive Plot Generator para obter os próximos eventos da história; e controlar o tempo gasto na simulação.

3. Interação e Responsividade

O modo de interação passo-a-passo, que já existia no Logtell, é mostrado na Figura 2, onde objetivos inferidos e eventos planejados são apresentados ao usuário ao fim de cada ciclo de simulação. O comando *continue* confirma o enredo até o ponto apresentado e solicita a continuidade do processo de simulação passo-a-passo. O comando *another* pede ao Logtell que faça um retrocesso e forneça outra alternativa para o passo de simulação recentemente terminado, mas ainda não confirmado. O uso desses comandos, além do fornecimento de ordens totais para dramatização dos eventos (uma vez que a simulação fornece apenas ordens parciais) correspondem a intervenções fracas do usuário no processo de geração do enredo.

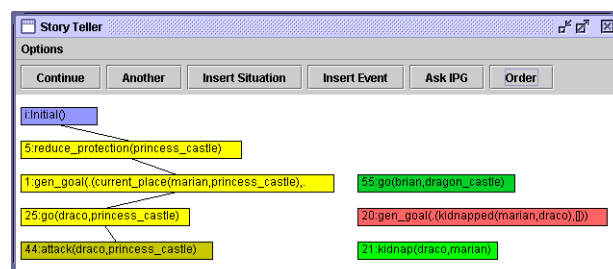


Figura 2. Interação passo-a-passo

Intervenções fortes podem ocorrer de duas maneiras. Uma delas é com o comando *insert situation*, que permite a especificação de objetivos a serem alcançados, ficando a forma como serão alcançados a cargo do IPG. Note que a situação inserida pode falhar caso não se ache nenhum plano possível ou se o esforço computacional exceder a configuração inicial do planejador. Um outro modo de se fazer interações fortes é a inserção explícita de eventos, usando o comando *insert event*. Assim como no caso anterior, o comando *continue* deve ser usado para validar as interações desejadas. O usuário pode também desfazer a intervenção, removendo os eventos ainda não incorporados e/ou rejeitados.

Em ambientes de alta responsividade, a interação não pode sempre exigir um alto nível de atenção, a menos que o usuário opte por pausar a dramatização para poder interagir. A interrupção da visualização da história, que era obrigatória na primeira versão do Logtell, ainda é permitida, mas é considerada um caso excepcional, sendo trocada pelo modo contínuo. Neste

modo, a história é gerada em tempo real conforme o usuário a assiste e interage com ela.

Nas várias instâncias do IPG em execução, não há somente a instância correspondente ao fluxo corrente de uma história, outras são usadas para evitar a interrupção da dramatização. A simulação precisa estar sempre alguns passos à frente da dramatização para manter a responsividade. Quando não há intervenção do usuário, os objetivos são inferidos, e os eventos, planejados, de forma contínua. Quando os usuários intervêm com o sistema, eles interagem de acordo com os eventos que estão sendo dramatizados. O Simulation Controller guarda “retratos” dos estados de cada simulação após cada ciclo, de forma que a simulação possa ser retomada a partir de um ponto anterior, possivelmente com outras alternativas de desencadeamento da trama, se isso for solicitado pelo usuário. A coerência lógica de uma intervenção requisitada é sempre verificada antes de ser incorporada, e descartada se inconsistente. Quando uma intervenção é incorporada, a simulação tem que descartar ciclos de simulação que foram previamente planejados e que não levaram a intervenção em conta. Para poder estar preparado para as intervenções, o sistema cria outras instâncias do IPG, simulando a incorporação destas intervenções a serem sugeridas como opções aos usuários. As intervenções só são sugeridas se a instância do IPG confirmar sua consistência. Se aceitas, os próximos eventos já terão sido planejados e haverá pouco risco de interrupção no fluxo da narração.

O tempo gasto para a simulação é constantemente monitorado pelo Simulation Controller. Quando há risco de interrupção na dramatização devido à falta de eventos já planejados, uma mensagem é enviada ao Drama Manager, de forma que políticas para estender a dramatização dos eventos correspondentes possam ser aplicadas. O Simulation Controller também tem um papel importante na implementação de novas formas de interação. Uma das suas tarefas mais importantes é criar os meios para a interação enquanto a geração dos enredos é feita e a sua dramatização ocorre no lado do cliente. Também no modo contínuo, as interações podem ser tanto fracas quanto fortes.

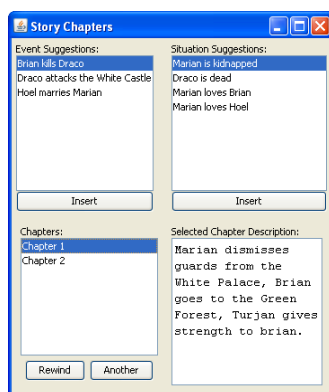


Figura 3. Interação contínua

Interações fracas basicamente circundam o fluxo “normal” da história, como se pode ter com os comandos *continue* e *another* do modo passo-a-passo. O Simulation Controller direciona o fluxo da história escolhendo automaticamente alternativas e a própria ordenação total dos eventos a serem dramatizados, de forma que o usuário, neste modo de interação, pode assistir à história sem ter que interferir. A idéia é que as histórias possam ser contadas mesmo se o usuário apenas assistir à dramatização, sem nenhuma intervenção. Até nesse caso, mantemos a possibilidade de assistir histórias diferentes porém coerentes e baseadas na mesma configuração inicial.

Para a obtenção de histórias variadas, o Simulation Controller seleciona aleatoriamente desencadeamentos de enredo gerados pela simulação. O resultado obtido através dos comandos *continue* e *another* no modo passo-a-passo ocorre, no modo contínuo, através de uma seleção aleatória de alternativas e da ordenações automática dos eventos de forma compatível com a ordem parcial gerada pela simulação. Os passos de simulação passam a ser capítulos gerados e apresentados continuamente. No modo passo-a-passo, ao fim de uma fase de simulação o usuário pode examinar os eventos ainda não incorporados. Pode também solicitar a dramatização repetidas vezes a partir de um certo ponto. No modo contínuo, como a história é apresentada sem interrupções, é preciso disponibilizar mecanismos para que o usuário possa voltar a pontos anteriores da história, seja para revê-los (comando *rewind*), seja para solicitar a geração de alternativas diferentes, como seria feito com o comando *another* no modo passo-a-passo. Para esse objetivo, diferentes retratos do processo de simulação ao fim de cada capítulo são guardados pelo Simulation Controller.

As interações fortes, como no modo passo-a-passo correspondem tanto à inserção de eventos específicos na trama quanto a diretivas de que certas situações devem ocorrer. No entanto, como o usuário está assistindo à história em paralelo, é desejável que possa intervir sem que grande esforço de concentração seja exigido. Para tal, o Simulation Controller gera automaticamente sugestões de eventos e situações a serem inseridas na história. O primeiro método para obter uma sugestão usa uma biblioteca de planos típicos, organizados como uma hierarquia de eventos básicos e complexos. Planos típicos geralmente consistem de certas combinações de eventos onde os vários personagens perseguem seus objetivos, mas também podem corresponder aos *motivos* [Aarne 1964] que são estruturas recorrentes compiladas através de estudos críticos sobre o gênero. O IPG contém um algoritmo para o reconhecimento de planos, baseado num algoritmo especificado por Kautz [Kautz 1991]. O procedimento é capaz de descobrir que alguns eventos são compatíveis com um motivo para o qual se tem um plano típico, deixando então que o Simulation Controller sugira a inclusão de eventos adicionais contidos neste plano. O segundo método é através

da análise das regras de inferência de objetivos em conjunto com a situação atual da narrativa. Através da análise, é possível detectar fatos a serem sugeridos que, uma vez que se configurem, podem levar à ativação de regras que provocam novos desencadeamentos para a história nos próximos capítulos. A Fig. 3 demonstra a interface de interação no modo contínuo. Eventos e situações são sugeridas através de listas que são continuamente atualizadas. Os capítulos também são listados e há uma janela onde os eventos do capítulo selecionado aparecem de forma resumida. Através do comando *rewind*, o usuário solicita o retorno da história à aquele capítulo. Através do comando *another*, é solicitado o retorno ao início do capítulo selecionado, mas adotando-se outra alternativa para o capítulo e o desenvolvimento da história a partir dessa alternativa. As *Event Suggestions* e as *Situation Suggestions* são as sugestões previamente citadas, que o usuário pode requisitar a inserção na história, para as próximas interações (capítulos).

Outras estratégias serão incorporadas para facilitar a interação com o usuário e tentar garantir a responsividade. Possibilidades incluem o **uso de conceitos abstratos** (permitindo a inserção de sugestões abstratas a serem resolvidas pela simulação); **controle de tensões narrativas** (manipulação de níveis de tensões como romance, velocidade; maior esforço para modelar o gênero [Polti 1945]); o já citado **modo multi usuário** (semelhante ao modo normal, porém usando votações pra escolher as sugestões inseridas na história); e **sugestões em linguagem natural** (igual às formas de interações já citadas, só forma de entrada seria outra).

No Logtell é usado um planejador não linear implementado em Prolog, adaptado e estendido a partir de [Yang et al. 1996], o qual tem grande flexibilidade para lidar com múltiplos objetivos inferidos e com as interações fortes dos usuários com a trama. Com a dramatização em paralelo, a geração dos enredos precisará ser rápida o suficiente para garantir a fluência da narrativa. Para solucionar o problema, pretende-se usar um algoritmo híbrido que combina o planejamento original com técnicas de HTN [Nau et al. 2003], de modo a reutilizar padrões típicos para a resolução de subproblemas.

4. Conclusão

A nova arquitetura e as estratégias buscam conciliar os requisitos de interatividade e responsividade, encontrados em um ambiente de televisão interativa, com a criação de história coerentes pelo Logtell. Apesar de serem focadas nas demandas da televisão digital, as soluções propostas são compatíveis com outras plataformas como a Web. Resultados obtidos até agora demonstram a viabilidade das estratégias que estão sendo incorporadas. Dentro do contexto da pesquisa, existem ainda outros tópicos que vêm sendo tratados tais como: a melhoria na dramatização,

buscando meios para controlar o tempo de duração das cenas e torná-las mais interessantes, com um número maior de variações; alternativas para o controle de câmera, que, no próprio cinema e na televisão fazem grande diferença no potencial dramático da história; e, por fim, a geração de texto para narração e diálogos, que têm grande impacto para o apelo dramático de qualquer história.

Referências

- AARNE, A. *The Types of the Folktale: A Classification and Bibliography*. Translated and enlarged by Stith Thompson, FF Communications, 184. Helsinki: Suomalainen Tiedekatemia, 1964.
- CAVAZZA, M., CHARLES, F. AND MEAD, S. Character-based interactive storytelling. *IEEE Intelligent Systems*, special issue on AI in Interactive Entertainment, 17(4):17-24, 2002.
- CIARLINI, A.E.M., POZZER, C. T., FURTADO, A.L. AND FEIJO, B. A Logic-Based Tool for Interactive Generation and Dramatization of Stories. In *Proc. ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, Valencia, 2005.
- GRASBON, D. AND BRAUN, N. A morphological approach to interactive storytelling. In *Proc. CAST01, Living in Mixed Realities. Special issue of Netzspannung.org/journal, the Magazine for Media Production and Inter-media Research*, p. 337-340, Sankt Augustin, Germany, 2001.
- KAUTZ, H. A.: A Formal Theory of Plan Recognition and its Implementation. In: *Allen, J. F. et al (eds.): Reasoning about Plans*. Morgan Kaufmann, San Mateo, 1991.
- MATEAS, M. AND STERN, A. Towards integrating plot and character for interactive drama. In: *Dautenhahn, K., editor, Socially Intelligent Agents: The Human in the Loop, AAAI Fall Symposium*, Technical report, p. 113-118, Menlo Park, CA, 2000. AAAI Press.
- NAU, D. S.; AU, T.-C.; ILGHAMI, O.; KUTER, U.; MURDOCK, W.; WU, D. AND YAMAN, F. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379-404, 2003.
- PAIVA, A., MACHADO, I. AND PRADA, R. Heroes, villains, magicians, ... Dramatis personae in a virtual story creation environment. In *Proc. Intelligent User Interfaces 2001*.
- POLTI, G. *Thirty-Six Dramatic Situations*. Whitefish, MT: Kessinger Publishing, 1945.
- RIEDL, M. AND YOUNG, R. M. An intent-driven planner for multi-agent story generation. In *the Proceedings of the 3rd International Conference on Autonomous Agents and Multi Agent Systems*, July 2004.
- YANG, Q., TENENBERG, J. AND WOODS, S.: On the Implementation and Evaluation of Abtweak. In: *Computational Intelligence Journal*, Vol. 12, Number 2, Blackwell Publishers (1996) 295-318.

Procedural Terrain Generation at GPU Level with Marching Cubes

Bruno José Dembogurski
MediaLab-UFF

Esteban W. Gonzalez Clua
MediaLab-UFF

Marcelo Bernardes Vieira
GCG-UFJF

Fabiana Rodrigues Leta
LMDC-UFF

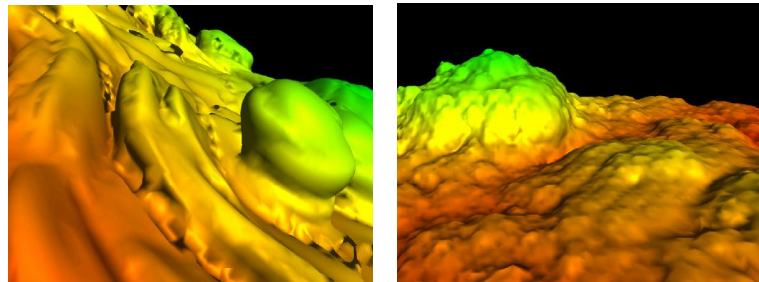


Figure 1: GPU procedural terrain generation

Abstract

This work presents a procedural terrain generation using the recent Marching Cubes Histogram Pyramids (also known as HPmarcher) implementation. Perlin Noise function is used to procedurally create the terrain. It runs entirely on the Graphics Processing Unit of Shader Model 3.0 and 4.0 graphics hardware.

Keywords: Marching Cubes, GPU, Procedural Generation, Terrains, Histogram Pyramids.

Authors' contact:

{bdembogurski, esteban}@ic.uff.br
marcelo.bernardes@ufjf.edu.br
fabiana@lmdc.uff.br

1. Introduction

Traditionally procedural terrains have been limited to height fields that are generated by the CPU and rendered by the GPU. However the serial processing architecture of the CPU is not suited to generating complex terrains which is a highly parallel task, Geiss [2007]. Another issue with height fields is the lack of interesting features such as caves and overhangs.

The GPU, on the other hand, is an excellent option for this kind of processing since it is a highly parallel device. GPUs of graphics cards are designed for large computational tasks with large requirement for memory bandwidth, based in massive parallelism.

Terrain creation is becoming the most costly element in video games and simulators, due the player high expectations for realistic virtual worlds. Procedural generation can create seamless terrains

with a realistic and natural look with a nice level of customization, without the need of huge amounts of disk space and loading time.

A voxel representation of the terrain allows much more features such as natural caves, tunnels, overhangs, cliffs without stretched walls and also allows a dynamic environment in a real time application.

This work presents a procedural terrain generation implementation using Histogram Pyramids Marching Cubes approach producing interesting terrains with a considerable amount of triangles achieving interactive frame rates.

2. Related Work

In the last years, GPU iso-surface extraction algorithms have been a topic of extensive research. In that field procedural terrain generation is also a well studied topic.

The main issue is that volumetric data consumes memory quite rapidly and the visualization of an extremely large and complex terrain can be a very challenging task. Another point to consider is the control over the terrain due the pseudo-random nature of this approach. Creating a specific type of terrain is a hard job usually requiring some additional technique to obtain the desired results.

Recently Geiss [2007] presented a good approach called *Cascades* to generate and visualize volumetric complex terrains using shader model 4.0 graphics hardware and new DirectX 10 capabilities such as the *geometry shader* (GS), stream output and rendering to 3D textures. It was able to create interesting and customizable terrains and also adding some nice features like particle system for waterfalls. However, this implementation has the

limitation of running only under Windows Vista since it is based on the DirectX 10 API.

The first use of the Histogram Pyramids algorithm running entirely on the GPU was presented by Zigler *et al.* [2007] for stream compaction in a GPU point listing generation algorithm.

Later Dyken and Ziegler [2007] presented a Marching Cubes approach extending the Histogram Pyramids structure to allow stream expansion, which provides an efficient method for generating geometry directly on the GPU. Currently, this approach outperforms all other GPU-based iso-surface extraction algorithms.

3. Marching Cubes

The Marching Cubes algorithm (MC) proposed by Lorensen and Cline [1987] is the most known and used algorithm for extracting an iso-surface from a scalar field.

Basically, in the MC algorithm, from a 3D grid of $N \times M \times L$ of scalar values a grid of $(N-1) \times (M-1) \times (L-1)$ cubic MC cells (voxels) is created. Each cell has a scalar value associated with each of its eight corners. The idea of the algorithm is “march” through all the cells, producing a set of triangles that approximates the iso-surface of that cell.

The topology of the iso-surface is defined by the classification of the eight corners of the MC cell as inside or outside the surface (being inside represented as “1” and outside as “0”). If all values in the cell corners are the same it is possible to discard that voxel, since it will be totally inside or outside the surface. If the corners have different values then the cell intersects the iso-surface.

According to that it is possible to determinate the voxel class using the following notation:

$$C^{i,j,k} = P_0, 2P_1, 4P_2, 8P_3, 16P_4, 32P_5, 64P_6, 128P_7$$

Suppose a corner $P_5^{i,j,k}$ is inside the iso-surface and all others are outside, according to the binary notation we will have the following MC class:

$$(P_0^{i,j,k}, \dots, P_7^{i,j,k}) = (0, 0, 0, 1, 0, 0, 0, 0),$$

That corresponds to the class 8 of a total of 256 possible classes, which can be reduced to 14 patterns due symmetry [Lorensen and Cline 1987]. The class also determinates which of the twelve edges are piercing the surface. If one end-point of the edge is inside the iso-surface and the other end-point is outside, that edge is intersecting the surface. For each of the 256 classes there is a corresponding triangulation of the edge intersections.

The position in the edge where the iso-surface intersects is obtained approximating the scalar field along the edge using a linear polynomial and finding the zero-crossing of that polynomial.

As presented in [Dyken and Ziegler 2007] the Marching Cube algorithm is implemented as a sequence of data stream compaction and expansion operations. The first stream operation determines the class of each voxel and check the triangulation table in order to obtain the number of triangles produced by each voxel. Discarding voxels that doesn't produce geometry generate the stream compaction. Each element in the output stream will be expanded according to the number of triangles determined by the triangulation table. The iso-surface is formed by connecting edge intersections on each element to form a set of triangles.

4. Histogram Pyramids

The core of the Marching Cubes implementation in this paper is the Histogram Pyramids algorithm [Dyken and Ziegler 2007], which is used to compact and expand data streams on the GPU. The structure of the Histogram Pyramid is identical to a MipMap where each level is a quarter size of the previous level pyramid but instead of taking the average of the elements to build the higher level, the algorithm sum the values.

In the input, the 3D voxel domain is mapped in the 2D domain (a large tiled 2D texture where each tile represents a slice of the voxel volume) to index a sequence of texture coordinates, which is known as a Flat 3D layout [Harris 2003]. The base layer have the number of elements to be allocated in the output stream, the rest of the pyramid is constructed following the MipMap reduction idea but adding the values of the 2×2 block in the texture. In the end, the top element will have the length of the output sequence.

All elements are extracted one by one descending into the sub-pyramids and inspecting the 4 elements in the corresponding 2×2 block until the base layer is reached. The stream compaction and expansion is relative to the number of elements that the input allocates in the output. For example, if one input element produces 4 elements in the output a stream expansion occurs. Otherwise, a stream compaction is performed if the input element produces no element in the output or a stream pass-through is performed when one input element produce one element in the output sequence.

5. Terrain Generation

Conceptually, the terrain surface can be described in a implicit form $f: \mathbb{R}^3 \rightarrow \mathbb{R}$. For any point given in the 3D space this function returns a single floating point value. Those vary over the space from positive and negative values where positive means outside and negative means inside the surface. Thus, one may choose a constant c for which the locus of points $f(p) = c$ form an isosurface defining the terrain.

The most common function used to create procedural terrains is the Perlin Noise function, which generates a natural look with no perceptive pattern.

The noise generation approach used in this work is similar to the one presented by Green [6]. This method implements the noise directly in the pixel shader instead of using pre-computed 3D textures. Green also uses a new interpolation function which is a Hermite polynomial of degree five resulting in a C^2 continuous noise function. It is also possible to use the original interpolation function which is less expensive to evaluate but has discontinuous second derivatives.

This approach has several advantages like less texture memory required and a higher quality interpolation. But the main enhancement in this approach is that it has a larger period, meaning that the noise patterns do not repeat often. This is essential for a natural and realistic look of the terrain.

Since the terrain can have an arbitrary size, it needs to be created in several passes because volumetric data consume memory really fast (e.g.: 256^3 voxels require a GeForce 8800GTX). The idea is to tile the scene using several cubes adding the appropriate coordinate's transformations when sampling the scalar field and extracting the geometry.

Considering a static terrain, it is possible to store the geometry in a buffer, since it is not need to sample and extract the geometry every frame. Using the Shader Model 4.0 hardware this is quite straightforward using the new *transform feedback* mechanism, which records vertex attributes of the primitives processed. The selected attributes can be written with each attribute in a separate buffer object or with all attributes interleaved into a single buffer object. If a geometry shader or program is active, the primitives recorded are those emitted by the geometry program. Otherwise, it will capture the primitives whose vertices are transformed by a vertex program or shader [5].

Basically, the general idea is to create a large buffer and incrementally fill this with the iso-surface each tile of the scene.

6. Results

All tests in this work were performed on a Nvidia GeForce 8800 GTS with 512mb ram on a Windows XP SP2 computer with an AMD64 2500+ CPU at 2.2GHz and 2GB of ram using the 175.16 version of the ForceWare (display driver).

The terrains in Figs. 2, 3 and 4 were generated with 127^3 voxels, 8 octaves for the turbulence function, a 2.0 lacunarity and a 0.5 gain for each iteration. These values can be changed for different results like a higher number of mountains or a more regular terrain. This experiment provided real time results with about 40.0 fps.

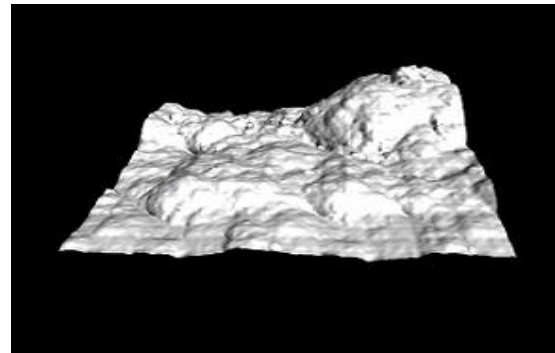


Figure 2: Terrain example with over 160,00 triangles and over 2 million voxels at a 38.2 fps.

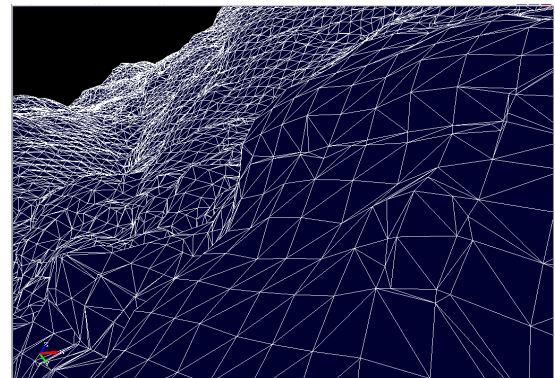


Figure 3: Wireframe representation of the same terrain.

It is also possible to manipulate the terrain creating flat spots in some areas (e.g. to support some building or any kind of construction in a game). For this, Geiss [2007] present a method to replace the density function with a flat spot within a linear radius of some point center. Other approaches like using a hand-painted texture and warping the coordinates to break the homogeneity of the terrain when using many octaves are possible ways to customize the terrain.

The Fig. 4 shows a terrain using a coordinate warping before applying the noise function. The result is an alien/organic look, showing the wide range of customization possibilities for this approach.

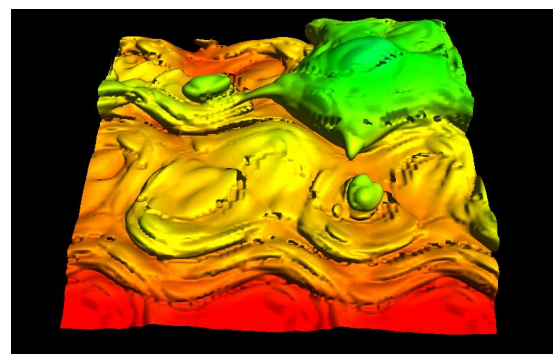


Figure 4: Alien look terrain using coordinate warping before applying the noise function.

Other volume dimensions were also evaluated for performance purposes: 31^3 , 63^3 and 255^3 . The relation between volume dimensions and the respective FPS values are shown in Fig 5. The frame rate depends on the number of triangles obtained the terrain.

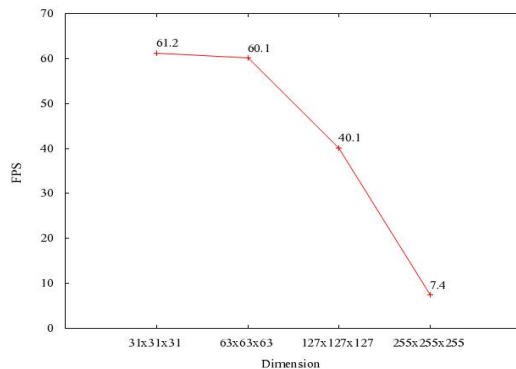


Figure 5: Dimension x FPS comparison

The FPS is related to the number of triangles presented by the terrain with a higher dimension volume, the FPS drop is expected since the number of triangles is about 5 times higher in each version. This graph shows that even with an extremely large number of polygons the visualization is possible and a very detailed terrain is shown in iterative frame rates. The low fps presented at 255^3 is related to the number of triangles used in this massive representation (in this case over 600.000 triangles were used).

The number of marching cubes cells processed per second is also a good way to evaluate the algorithm effectiveness (Fig. 6). One may see that even with cubic time/space complexity, the histogram pyramid approach is a powerful isosurface extraction algorithm. It performs fast with larger datasets and is well suited for this non restrictive terrain representation.

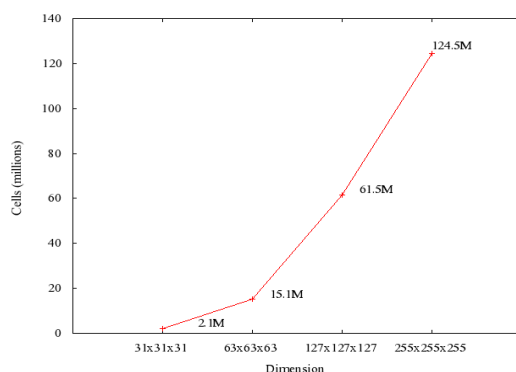


Figure 6: Dimension x Cells processed relation

7. Conclusion and Future Work

This work presented a fast procedural terrain generation using the Histogram Pyramids

Marching Cubes approach. The terrain is generated with a Perlin noise function to create a scalar field. This strategy provided good results with interactive frame rates and customizable features.

The performance of the presented approach was evaluated in function of grid dimensions. The results are promising since the implementation achieved a real time frame rate with 127^3 voxels.

As a future work, we intend to extend the Histogram Pyramids approach with a CUDA framework. The voxel classification and the pyramid construction pass can be enhanced because CUDA supports 2D buffer texture sampler.

We also intend to improve the terrain with texturing, shading and some kind of erosion (thermal or hydraulic). Texturing is a challenge to procedural generations due to its arbitrary topology but one solution is to apply three different planar projections one along each primary axes with some blending between areas [Geiss 2007].

References

- EBERT D.S *et al*,” *Texturing & Modeling. A Procedural Approach*”, 3rd ed, Morgan Kaufmann Publishers, 2003.
- DYKEN, C., ZIEGLER, G. “High-speed Marching Cubes using Histogram Pyramids”. *Proceedings of EUROGRAPHICS 2007*, Volume 26, Number 3.
- GEISS, R. “Generating Complex Procedural Terrains Using the GPU”. *GPU Gems 3*. Chapter 1, pp. 7-37. 1st Ed. 2007.
- GREEN, S., “Implementing Improved Perlin Noise”, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation.*, 2005, pgs 409-415.
- HARRIS M. J., III W. V. B., SCHEUERMANN T., LASTRA A.: *Simulation of cloud dynamics on graphics hardware*. *Proceedings of Graphics Hardware 2003*.
- LORENSEN W., CLINE H. E.: *Marching cubes: A high resolution 3d surface construction algorithm*. *Computer Graphics (SIGGRAPH 87 Proceedings)* 21, 4 (1987), 163–170.
- OLSEN, J. “Real-time Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games”, IMADA University of Southern Denmark, 2004.
- ZIEGLER G., TEVS A., TEHOBALT C., SEIDEL H.-P.: “GPU Point List Generation through Histogram Pyramids”. *Tech. Rep. MPI-I-2006-4-002*, Max-Planck-Institut für Informatik, 2006.
- OLSEN, J. “Real-time Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games”, IMADA University of Southern Denmark, 2004.

Arquitetura de Servidor de Xadrez sobre XMPP com Cliente Web

Rubens Suguimoto
Eduardo Ribas

Gabriel Ramos
Danilo Yorinori

Raphael Ribas
Luis Bona

Fabiano Kuss
Fabiano Silva

Ulysses Bonfim
Alexandre Direne

Pedro Rocha

Resumo

É apresentada a arquitetura do servidor *Xadrez Livre*, a qual se baseia no protocolo XMPP. Os clientes são páginas Web para evitar a instalação de software adicional. Também estão destacados no texto os detalhes de implementação do servidor voltados às necessidades de ensino do projeto de âmbito nacional intitulado "Apoio Computacional ao Ensino de Xadrez nas Escolas" e financiado pelo Ministério da Educação. Aspectos de desempenho do servidor e as metas futuras de pesquisa finalizam o texto.

Keywords:: Xadrez, Arquitetura, XMPP, Cliente Web, Software Livre

Author's Contact:

C3SL - Departamento de Informática - UFPR,
Curitiba-PR, 81.531-980.
E-mail: xadrez-devel@c3sl.ufpr.br

1 Introdução

Atualmente existem milhões de jogadores de xadrez ao redor do mundo. Federações estaduais e nacionais promovem torneios, circuitos e olimpíadas de xadrez, onde os melhores jogadores são premiados. Além disso, jogadores de alto nível recebem títulos de acordo com sua perícia e desempenho no jogo, concedidos pela Federação Internacional de Xadrez, em boa parte de maneira remota pela Internet.

Com a popularização da Internet surgiram alguns servidores de xadrez que permitem a pessoas localizadas em qualquer ponto do mundo jogarem xadrez via web utilizando um tabuleiro eletrônico. Apesar de qualquer usuário da Internet poder se conectar, as arquiteturas utilizadas acabavam limitando o acesso quase que somente a especialistas em xadrez e em informática. A interface com o usuários era por linha de comando e requeria grande conhecimento sobre regras utilizadas na interação. Além disso, era necessária a instalação de software adicional para a visualização gráfica do tabuleiro.

Com o surgimento da Internet 2.0, foi possível elaborar uma arquitetura de servidor de xadrez mais flexível e que privilegia diferentes contextos de utilização. A arquitetura apresentada neste artigo contempla todos os componentes para a realização de jogos de xadrez via internet. Ela foi desenvolvida inicialmente para o contexto educacional mas pode ser utilizada em vários outros contextos com exatamente o mesmo servidor, substituindo-se somente os componentes de interface com os usuários.

A implementação foi desenvolvida como parte do projeto intitulado "Apoio Computacional ao Ensino de Xadrez nas Escolas". O projeto teve início no final do ano de 2006, constituído como uma parceria entre a Secretaria de Educação a Distância do Ministério da Educação (MEC/SEED), a Universidade Federal do Paraná (UFPR) e o Centro de Excelência em Xadrez (CEX). Assim, o componente de interface com o usuário denominado ambiente de jogo *Xadrez Livre* foi desenvolvido para ser utilizado no contexto educacional, cujo acesso está disponível em <http://xadrezlivre.c3sl.ufpr.br>

O artigo está estruturado em 7 seções. Depois desta introdução, são apresentados os trabalhos correlatos, com a descrição das arquiteturas de servidores de xadrez utilizadas em outros trabalhos. Na seção conceitos relevantes, as tecnologias e os conceitos utilizados neste trabalho são descritos sucintamente. Na quarta seção, a arquitetura geral do servidor de xadrez proposta neste trabalho é apresentada, sem entrar em detalhes de implementação. A seção implementação disponível, descreve os detalhes da implementação

dos componentes da arquitetura proposta, as dificuldades que apareceram no decorrer do desenvolvimento e a trajetória do projeto. Na sexta seção, são apresentados os resultados dos testes de desempenho da implementação disponível. E, por último, as conclusões.

2 Revisão da literatura

Lee Newton [Newton 2004] descreve o desenvolvimento de jogos multiplayer utilizando o Jabber como meio de comunicação. A idéia de Lee é utilizar aplicativos em Flash como cliente e servidor ao mesmo tempo que recebem mensagens de outros clientes, processam e respondem. Em seu trabalho ele descreve alguns problemas que ocorreram ao implementar o Jabber em seus aplicativos e suas soluções.

O FICS [FICS 2008], Free Internet Chess Server, é uma implementação de servidor de xadrez em software livre. Os clientes se conectam diretamente no servidor através de linhas de comandos. Devido à dificuldade de usuários inexperientes em aprender os comandos, atualmente existem várias interfaces gráficas que emulam um tabuleiro gráfico. No entanto, para o usuário usar esse tabuleiro gráfico, é necessário instalar algum software adicional. Dentro desse contexto do FICS existe uma implementação brasileira com menos funcionalidades, o ChessD [ChessD 2008].

Netto [Netto et al. 2005] propõe a criação de um ambiente, apoiado em recursos ofertados pela Web utilizando as ferramentas clássicas tais como fóruns, chats e murais para exercerem apoio à aprendizagem virtual. Neste ambiente os usuários de variados níveis desempenham atividades básicas do xadrez. No trabalho é proposto um sistema colaborativo multiusuário, fazendo a identificação de vários atores e seus papéis no sistema, destacando a tarefa do sistema além de descrever a implementação e os processos colaborativos envolvidos no processo de ensino-aprendizagem do xadrez. O sistema proposto utiliza a linguagem de programação Java na concepção da interface. As páginas utilizam servlets JSP para promover integração com a Internet e banco de dados MySQL para persistência das informações.

O ChessPark [ChessPark 2008] utiliza uma interface web e usa o XMPP com o protocolo "JEP-xxxx:Chess Game"[XMPP 2006a] que define as mensagens relacionados ao jogo de xadrez. Acredita-se que o ChessPark utiliza as mesmas abordagens usadas nesse trabalho. No entanto, por ser um projeto proprietário muitas das informações de implementação não são divulgadas, o que dificulta fazer uma comparação.

Wang e Liuz [Wang1 and Liu1 2006] mostraram em seus trabalhos a arquitetura de servidores descentralizado para jogos de xadrez chinês utilizando o Jabber para obter alta escalabilidade, melhor desempenho, segurança e tolerância a falhas. A idéia seria a de utilizar o Jabber para interligar os jogadores, sendo que estes poderiam criar seus próprios servidores de jogo de forma que eles estariam conectados a um banco de dados que guarda as pontuações. A implementação apresentada, mostra que o cliente é um software que precisa ser executado na máquina do usuário.

3 Conceitos relevantes

Os conceitos mais gerais, relacionados à arquitetura proposta, são XML e XMPP. Já HTTP, Bosh, Apache, Javascript/DOM, AJAX, HTML, CSS e XHTML estão relacionados à implementação da arquitetura no projeto. Todos os conceitos estão descritos neste tópico.

3.1 XMPP

XMPP é acrônimo de eXtensible Messaging and Presence Protocol e é um protocolo definido e mantido pela XMPP Standards Foun-

dation para ser utilizado no envio de mensagens instantâneas e é baseado no XML. Para o presente trabalho, foram utilizados os seguintes documentos do protocolo XMPP: (a) Core [Jabber 2004a]; (b) Instant Message and Presence [Jabber 2004b]; (c) Multi-user chat [XMPP 2008].

O primeiro define como é o formato das mensagens XMPP e como a conexão é realizada. O Segundo está relacionado às mensagens instantâneas trocadas pelos usuários e às mensagens de registro de presença dos usuários no ambiente e consequentemente no servidor. O terceiro explica como utilizar os recursos de salas de conversação.

3.2 Bosh

Bosh é acrônimo de Bidirectional-streams Over Synchronous HTTP, ou Fluxo Bidirecional Sobre HTTP Síncrono. O Bosh é um protocolo definido também pelo XMPP Standards Foundation e nele está definido como fazer o transporte de XML de forma rápida e eficiente sobre o HTTP. Esse transporte é feito de forma bidirecional, tanto do cliente para servidor como do servidor para o cliente.

O Bosh está definido no XEP-0206 [XMPP 2006b], e nele está definido todo mecanismo de abertura de conexão (sessão) e também meios de mantê-las.

3.3 Apache

O Apache [Apache 2008] é um servidor HTTP livre desenvolvido e mantido pelo The Apache Software Foundation e ele gerencia todas as requisições dos clientes.

3.4 Javascript e DOM

Javascript é linguagem interpretada implementada na maioria dos navegadores e segue o padrão especificado pelo ECMA-262 [ECMA]. A linguagem é utilizada para processar páginas web de forma que a torne mais dinâmica e interativa.

Dentro do Javascript, há a implementação do DOM [W3C 2008], Document Object Model ou Modelo de Objetos de Documento. O DOM oferece mecanismos para acessar os elementos de um documento de forma separada. Nesse contexto de linguagem web, o DOM foi implementado sobre os elementos HTML de forma que fica mais simples acessar os elementos da página e modificá-los dinamicamente.

3.5 AJAX

Acrônimo de Asynchronous JavaScript and XML, o AJAX é um mecanismo usado para obter dados do servidor utilizando o objeto XMLHttpRequest. O AJAX funciona em "background" de forma que não altera o conteúdo da página que está sendo mostrada no navegador.

Atualmente, o AJAX não tem um padrão bem documentado de seu uso. Por isso, ao contrário do que mostra o nome, o AJAX pode funcionar com transmissão de dados que não estão em formato XML, não precisa ser assíncrono e também não precisa utilizar o Javascript.

3.6 XHTML

Acrônimo de eXtensible Hypertext Markup Language. O XHTML [W3C 2002] é uma forma padronizada de escrever o HTML seguindo a mesma estrutura rígida do XML. Assim como segue a estrutura do XML, a utilização de folhas de estilo(CSS) para dispor os elementos é altamente recomendada.

A flexibilidade e a redução de tamanho dos arquivos a serem transmitidos, tanto XHTML quanto o CSS, são fatores encorajadores ao uso do CSS.

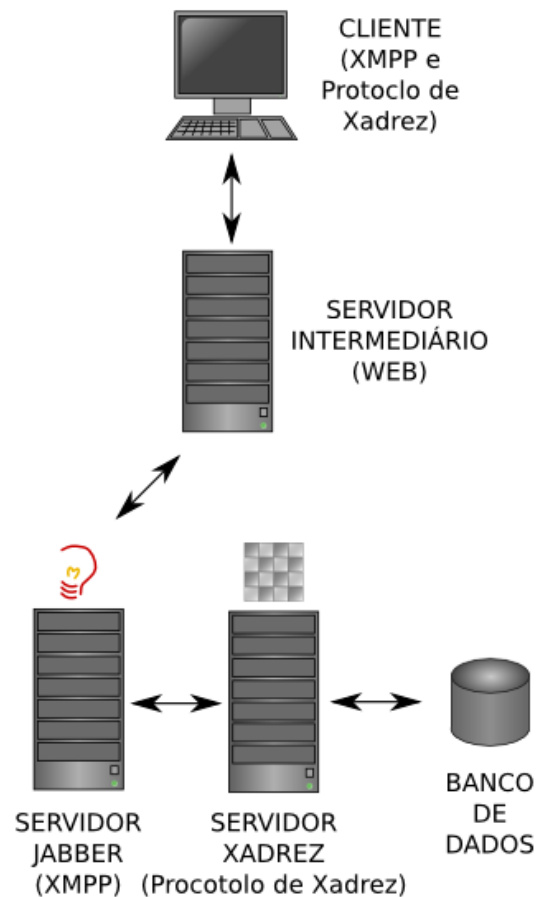


Figura 1: Arquitetura geral

4 Arquitetura

A arquitetura utilizada é baseada no modelo cliente-servidor e está dividida em cinco partes: cliente web, servidor intermediário, servidor jabber, servidor de xadrez e o banco de dados. A Figura 1 apresenta a visão funcionalista adotada. Como a arquitetura está sendo implementada sobre o Jabber como componente principal, todas as mensagens e informações que trafegam são baseadas no protocolo XMPP. Nas subseções seguintes serão descritos os quatro componentes mostrando apenas o que cada um desses componentes representam dentro da arquitetura geral, sem entrar em muitos detalhes de implementação.

4.1 Cliente Web

O cliente se conecta no servidor intermediário e este com o Jabber através de mensagens bem definidas no protocolo XMPP. O cliente contém mecanismos para identificar, interpretar, enviar e, se for necessário, responder às mensagens recebidas.

Um outro ponto importante no cliente é a forma de mostrar os dados embutidos nas mensagens. Apesar de um ser humano conseguir interpretar dados no padrão XMPP, se forem enviadas várias mensagens em pequenos intervalos de tempo, essa tarefa acaba se tornando impraticável. Lembrando também que a interface com o usuário pode ser um diferencial para atrair e manter os usuários no sistema.

4.2 Servidor Intermediário

Devido à implementação atual dos navegadores, é necessário existir um serviço intermediário que faz a conexão entre o cliente e o servidor Jabber. Esse serviço intermediário pode ser considerado o servidor Web, o qual responde pela comunicação com o cliente

web e do cliente web para o servidor Jabber. O caminho inverso da comunicação também é válido.

4.3 Servidor Jabber (XMPP)

O servidor Jabber é uma implementação do protocolo XMPP. Ele é responsável por toda a comunicação e também oferece funcionalidades de conversa instantânea, perfil de usuário e lista de contatos. O Jabber também contém um mecanismo de extensão em que é possível ligar componentes que aumentam a funcionalidade do sistema sem que haja necessidade de alterar o código fonte.

Na arquitetura apresentada, o Jabber seria responsável por gerenciar os clientes, estabelecendo a comunicação entre eles, além da comunicação cliente-servidor. O servidor *Xadrez Livre* adota formatos de mensagens chegam ao seu destino sem problemas de alterações ou inconsistências de seus conteúdos.

4.4 Servidor de Xadrez

O *Xadrez Livre* é um programa que funciona como um componente do Jabber que basicamente recebe requisições dos clientes e retorna uma resposta. Como o foco do componente é o jogo de xadrez, o servidor fica responsável por organizar as partidas, torneios, armazenar ratings e as funções administrativas relacionadas ao xadrez.

Atualmente não existe um protocolo XMPP oficial para jogos de xadrez. Então o servidor que foi implementado dessa maneira teria que desenvolver um novo padrão ou utilizar algum protocolo já existente não oficial. O protocolo terá que conter todas as mensagens que estão relacionados ao jogo de xadrez, como por exemplo, mensagens para o jogador fazer um movimento e o servidor enviar um novo estado do tabuleiro para o cliente.

4.5 Banco de dados

O *Xadrez Livre* possui um banco de dados próprio, diferente do banco de dados do Jabber. Esse banco de dados é responsável por armazenar itens exclusivos do servidor. Exemplos de dados são: (a) o rating de um usuário; (b) os torneios como grupos planejados de partidas; (c) as partidas já realizadas por meio do servidor, aleatórias ou planejadas em um torneio; (d) lances individuais e granulados de cada partida; (e) o tipo explícito dos usuários com suas permissões de acesso. Muitos outros dados existem.

5 Implementação disponível

Atualmente existe uma implementação de toda essa arquitetura integrada a um cliente web no site "http://xadrezlivre.c3sl.ufpr.br". Dentro desse site o usuário pode usufruir de funcionalidades de conversas e, principalmente, a parte de jogo.

Todos os serviços e o banco de dados foram implementados na mesma máquina, devido ao recurso limitado de hardware e também por facilitar a administração. O cliente web, o servidor Intermediário, servidor Jabber, servidor de Xadrez, o banco de dados e o protocolo de xadrez usados na implementação estão descritos nas subseções.

5.1 Cliente Web

O cliente é uma página web que contém uma interface implementada em XHTML, AJAX e o Javascript/DOM. O código XHTML (HTML e o CSS) é responsável por mostrar elementos e acertar seus estilos dentro da página de forma coerente ao trabalho de design e layout feito pela equipe.

Uma das restrições ao utilizar HTML é a obrigação de transmitir uma nova página sempre que houver necessidade de comunicação. Isso piora a interface do usuário pois demanda um maior tempo de espera entre cada ação devido à quantidade de dados a serem transmitidos e processados. A alternativa é fazer a comunicação utilizando **AJAX**, que possibilita o processamento assíncrono com o servidor. Em outras palavras, é possível trocar informações sem precisar atualizar a página inteira a cada interação.

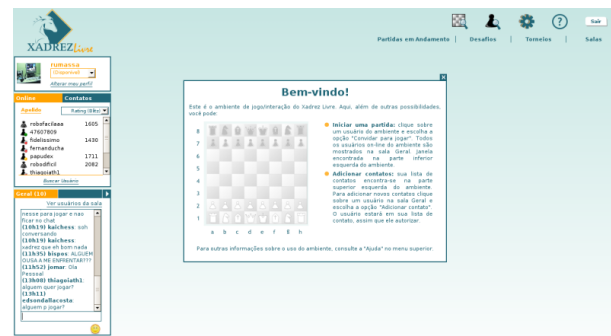


Figura 2: Tela de um usuário que acaba de entrar no ambiente

Os códigos Javascript/DOM são responsáveis por toda a dinâmica da página, atualizando, gerenciando e mostrando os dados para o usuário dentro do navegador web apenas alterando ou criando alguns elementos HTML. Através do Javascript é possível ativar o AJAX e obter os dados do servidor, em seguida enviar, receber, processar e mostra os dados.

Todas as combinações dessas ferramentas fazem com que a interface web seja capaz de enviar e receber mensagens do servidor, interpretá-las e em seguida mostrar ao usuário os dados obtidos. Em outras palavras, a interface oferece um ambiente interativo para o usuário e limita suas ações dentro do sistema. A Figura 2 mostra o usuário logado no ambiente de jogo.

5.2 Servidor Intermediário

Na arquitetura deste artigo, o Bosh serve para fazer a comunicação das mensagens enviadas pelo navegador, através do Ajax, passando pelo Apache até o servidor Jabber. Atualmente, os posts HTTP não garantem a persistência da conexão. Ela pode não ser mantida durante posts realizados por um mesmo cliente, ou seja, não necessariamente posts consecutivos serão enviados pela mesma conexão TCP. Devido a essa restrição, houve a necessidade de criar um servidor intermediário que simule uma conexão persistente.

Para proporcionar uma conexão bidirecional, deve ficar garantido que o cliente e o Bosh mantenham uma requisição pendente, mesmo não precisando se comunicarem por um tempo. Caso o tempo tenha excedido, ele responde com uma mensagem vazia ou mensagem de espera. Se o servidor Jabber mandar alguma mensagem, o BOSH a envia para o cliente na requisição pendente que ele mantém. Quando o tempo limite de uma requisição que o BOSH mantém é excedido, o cliente deve mandar outra. Caso o BOSH não receba mais uma requisição do cliente dentro do tempo especificado, ele o considera desconectado, informa sua desconexão ao Jabber e fecha a conexão referente a este cliente.

5.3 Servidor Jabber

O servidor Jabber é uma implementação do protocolo XMPP e uma das principais funcionalidades do Jabber dentro da arquitetura é prover toda a comunicação feita dentro do sistema. O Jabber também prove as funcionalidades de conversas, salas, lista de contatos e funções administrativas que liberam o servidor de xadrez de cuidar das conversas e administração dos usuários.

Em nossa implementação usamos o EJabberD [EJABBERD 2008], software livre, que contém boa parte do XMPP implementado em Erlang. A implementação do EJabberD oferece todas as funcionalidades necessárias para o projeto do Xadrez Livre, as mesmas descritas no parágrafo anterior.

5.4 Servidor de Xadrez

O servidor de Xadrez desenvolvido é uma extensão do Jabber que se conecta como um componente e é escrito em C++. O servidor tem acesso ao banco de dados através de uma API do Postgre. O

componente também é responsável por tratar de todas as mensagens definidas no protocolo de xadrez. Antes de iniciar uma partida, um jogador precisa enviar um convite a outro e a partida se inicia quando o outro jogador aceitar o convite. Todo esse processo de convite e de iniciar uma partida é gerenciado pelo servidor. O servidor também oferece funcionalidades de buscar partidas antigas e observar partidas em andamento.

As funções administrativas são acessíveis por usuários que receberam privilégios de outros administradores. Muitas das funções administrativas de desconectar usuários e banir já se encontram no Jabber, mas outras tiveram que ser implementadas no servidor devido ao caso específico do xadrez. As funções de adjudicar partida (definir vencedor nas partidas suspensas) e escolher tipo de usuário são funcionalidades específicas.

6 Teste de desempenho

Foram feitos alguns testes de desempenho utilizando robôs, que faziam movimentos simples. Todos os robôs foram executados em rede local e os servidores contidos na mesma máquina. Os testes tiveram o objetivo de verificar apenas latência por número de usuários e a carga da CPU por número de usuários.

A máquina onde está os servidores tem a configuração de 2 Dual-Core AMD Opteron(tm) Processor 2220 (2.8Ghz) com memória de 8 Gb, com Linux Kernel 2.6.22 64 bits, instalada em rede de 1 gigabit Ethernet. O tempo de cada teste foi de 10 minutos com o número de usuários variando entre 500, 1000, 1500, 2000, 2500, 3000 e 3500. As Figuras 3 e 4 mostram os gráficos que representam a latência média das mensagens (em segundos) por número de usuário e o a média da carga da CPU por número de usuários.

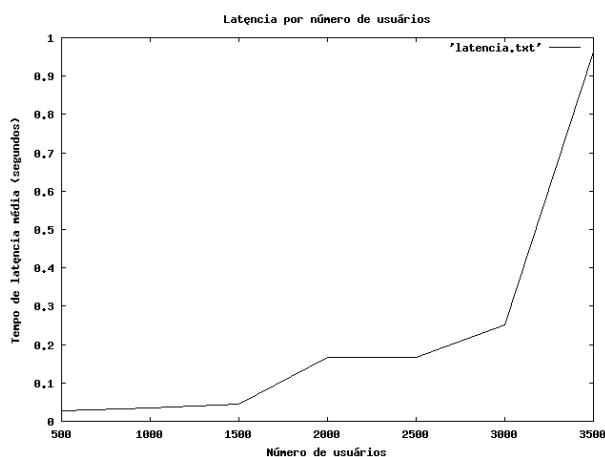


Figura 3: Latência por número de usuários

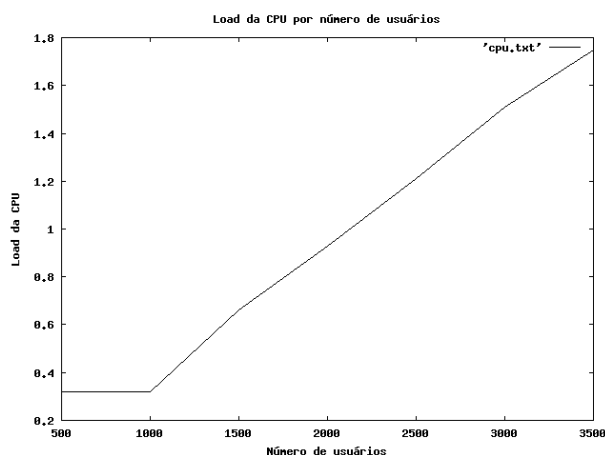


Figura 4: Carga da CPU por número de usuários

A latência com 3500 usuários ficou alta. Isso se deve ao fato da máquina onde estavam os robôs ter chegado ao seu limite. A carga da CPU foi aceitável durante testes. Maiores problemas só ocorreriam se o valor chegasse a 4.0, caso em que os processos dos servidores ocupariam 100% dos processadores.

7 Conclusões

A arquitetura com cliente web pode ser utilizada para o desenvolvimento de outros jogos de forma a manter a interatividade. Com esse método o cliente não precisaria instalar nenhum software adicional. A utilização do Jabber (protocolo XMPP) como meio central de comunicação é outro ponto importante. Ele mesmo provê uma camada transparente de comunicação ao desenvolvedor do componente de jogo.

Como a implementação disponível é toda software livre, o código fonte será em breve disponibilizado no SourceForge para qualquer pessoa interessada no projeto. Além disso, a arquitetura proposta permite a substituição do componente de interface com o usuário para ser utilizado em contextos diferentes da implementação disponível, que é o contexto educacional.

Referências

- APACHE, T. A. S. F., 2008. Apache http server project. <http://httpd.apache.org/>, ago.
- CHESSD, 2008. Chessd. <http://chessd.sourceforge.net/index-en.php>, ago.
- CHESSPARK, C. P., 2008. Chess park. <http://www.chesspark.com/about/>, ago.
- ECMA, E. I. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
- EJABBERD, 2008. Ejabberd. <http://www.process-one.net/en/ejabberd/>, ago.
- FICS, F. I. C. S., 2008. Free internet chess server. <http://www.freechess.org/>, ago.
- JABBER, J. S. F., 2004. Extensible messaging and presence protocol (xmpp): Core. <http://www.xmpp.org/rfc/rfc3920.html>, out.
- JABBER, J. S. F., 2004. Extensible messaging and presence protocol (xmpp): Instant messaging and presence. <http://www.xmpp.org/rfc/rfc3921.html>, out.
- NETTO, J. F. M., TAVARES, O., AND MENEZES, C. 2005. Um ambiente virtual para aprendizagem de xadrez. In *Workshop de Jogos Digitais na Educação (SBIE-2005)*, SBC, Juiz de Fora.
- NEWTON, L. 2004. Jabber for multiplayer flash games. *Computers in Entertainment (CIE)* 2, 1 (Oct.), 13–13.
- W3C, 2002. Xhtml2 working group home page. <http://www.w3.org/MarkUp/>, ago.
- W3C, 2008. Document object model (dom). <http://www.w3.org/DOM/>, ago.
- WANG1, Q., AND LIU1, S. 2006. Chinese chess based on jabber. *Technologies for E-Learning and Digital Entertainment 3942/2006*, 1 (mar), 706–710.
- XMPP, X. S. F., 2006. Jep-xxxx: Chess game. <http://www.xmpp.org/extensions/inbox/chess.html>, set.
- XMPP, X. S. F., 2006. Xep-0206: Xmpp over bosh. <http://www.xmpp.org/extensions/xep-0206.html>, jun.
- XMPP, X. S. F., 2008. Xep-0045: Multi-user chat. <http://www.xmpp.org/extensions/xep-0045.html>, ago.

RPG Educativo 3D Utilizando Linguagem de Script e Agentes

Rudimar L.S. Dazzi, Benjamim G. Moreira, Edmar Souza, *Heitor Adão Jr

UNIVALI - Universidade do Vale do Itajaí
CTTMar – Centro de Ciências Tecnológicas da Terra e do Mar
LIA - Laboratório de Inteligência Aplicada

Abstract

O RPGEDU é um jogo didático do tipo Role Playing Game (RPG) usando a tecnologia 3D. O projeto engloba diferentes conteúdos de 5ª a 8ª séries, sendo apresentados em forma de atividades pedagógicas espalhadas pelo jogo. Para dar suporte as atividades pedagógicas foi desenvolvida uma biblioteca para fazer interface com uma linguagem de script, permitindo assim construir uma aplicação com conteúdo flexível e facilmente extensível. A biblioteca ainda possui suporte a arquivos XML, o que permitiu maior independência do motor principal. A estrutura de scripts também foi reaproveitada para desenvolver as ações dos NPCs e os scripts das fases do jogo.

Keywords: RPG, Jogos educacionais, Agentes.

Authors' contact:

{rudimar, benjamin, edmar.souza}@univali.br
*heitoradao@gmail.com

1. Introdução

Cada vez mais se busca uma maior versatilidade na educação, tanto no sentido de horário como de local para o estudo e aprendizado, e é nesse ponto que a informática e a tecnologia em geral fazem o seu papel, ajudando a alcançar os alunos independente de local e de horário que possuem disponível.

O grande propulsor disso foi o crescimento da Internet, pois através dela os estudantes podem ter acesso a informações em qualquer lugar, como, por exemplo, sua casa ou lugar de trabalho, sem precisarem deslocar-se até um local específico, e como essas informações podem ser acessadas em qualquer horário, possibilita que o aluno estude no horário que desejar.

De acordo com [Franco 2000], pesquisadores que investigam o uso de computadores na educação alegam que a informática possui uma ação positiva para o desenvolvimento da capacidade cognitiva e provoca um rompimento da relação vertical entre alunos e professor da sala de aula tradicional, fazendo do aprendizado uma experiência mais cooperativa. As radicais transformações da informática nos anos noventa reforçaram ainda mais a adoção dessa tecnologia nos meios educacionais.

O ensino por meio do computador implica que o aluno, através da máquina, possa adquirir conceitos sobre praticamente qualquer assunto. É possível

dividir os softwares educacionais segundo sua forma de abordagem em duas categorias: abordagem por instrução explícita e direta e com abordagem de exploração auto-dirigida [Valente 2002].

Jogos educacionais é um grupo da abordagem de exploração auto-dirigida e os defensores desta filosofia pedagógica acreditam que a criança aprende melhor quando é livre para descobrir relações por si só, ao invés de ser explicitamente ensinada [Valente 2002].

Os jogos educacionais apresentam um ambiente com um modelo de simulação onde o tipo de ação executada pelo aluno faz diferença no resultado do jogo, ou seja, o aluno não tem uma participação estática durante a aprendizagem, fazendo somente o papel de receptor de informações, ele participa ativamente interagindo com o ambiente.

Ao desenvolver um jogo educativo é necessário levar em consideração quesitos que em jogos sem objetivo educacional muitas vezes são deixados de lado. De acordo com [Furtado, Santos e Gomes 2003], pela necessidade de unir diversão com aprendizado os jogos educacionais se tornam um desafio complexo em relação à aceitação final por parte do usuário, sendo o maior desafio o de encontrar o ponto de equilíbrio entre diversão e aprendizado, para que um não prejudique o outro.

Dentre os estilos de jogos disponíveis está o RPG. Segundo [Kumick 2003], o RPG é uma excelente ferramenta educacional por possuir algumas características importantes para educação como promover a socialização, a cooperação e a criatividade, além de ser totalmente interativo e poder abranger várias disciplinas em um único jogo. Um assunto muito discutido na educação é como aumentar a motivação dos alunos, e assim buscar uma melhora nos índices de aprendizagem e também evitar a evasão escolar. O uso de jogos como estratégia de ensino é extremamente eficaz para o aumento da motivação dos alunos [Tanaka 2005].

Os jogos nem sempre trazem resultados positivos para a educação. Alguns jogos podem promover, além da motivação e aquisição de conteúdo, a competitividade excessiva. Então uma possível solução é o uso de jogos que também promovam o cooperativismo e socialização [Tanaka 2005].

Analisando as características necessárias para que um jogo traga somente bons resultados na educação e as características do RPG, pode-se notar que o RPG se encaixa perfeitamente a essas necessidades, tornando-se assim um tipo de jogo recomendado para o uso na educação.

O RPGEDU é um jogo didático do tipo RPG em terceira dimensão. Esse projeto tem como público alvo os alunos de 5ª a 8ª séries do ensino fundamental e conteúdos pedagógicos de várias disciplinas diferentes.

No desenvolvimento do projeto, utilizou-se a engine Ogre3D para a renderização de imagens, Crazy Eddie's GUI System (CEGUI) para o desenvolvimento da interface com o usuário, a biblioteca OPAL para física e detecção de colisão, a biblioteca OpenAL para a sonoplastia, a linguagem C++ no desenvolvimento do núcleo do jogo e a linguagem de script LUA, com o auxílio da biblioteca LuaBind, para o desenvolvimento das atividades pedagógicas e implementação do comportamento dos *Non Player Characters* (NPC).

2. Metodologia

O objetivo da proposta foi construir um jogo de computador que englobasse a idéia de um RPG e diferentes conteúdos de 5ª a 8ª séries. O que se propôs então foi um jogo que atenda as necessidades desses alunos, sendo que o professor determinará as atividades que deverão ser desenvolvidas, bem como o tempo que os alunos poderão jogar.

O projeto teve início com conteúdo totalmente estático, exclusivamente programado em linguagem C++. Cenários e atividades pedagógicas estavam programados diretamente no *engine*, o que tornava qualquer alteração difícil e demorada devido ao tempo causado pela re-compilação.

Para mudar esse cenário de desenvolvimento primeiramente foram desenvolvidos testes utilizando a linguagem de script Ruby, adaptando com sucesso uma atividade pedagógica do RPGEDU usando esta linguagem. Mas por fim, foi utilizada a linguagem de script Lua onde esta foi integrada com o motor principal e foram reescritas as atividades pedagógicas totalmente usando Lua, para deixar o motor mais flexível.

Este trabalho tem como objetivo descrever o uso de scripting para uso com atividades pedagógicas e com agentes não jogáveis controlados por computador, os *Non Player Characters* ou NPCs.

2.1 Linguagem de script

Linguagens de script são linguagens de programação desenvolvidas com o objetivo de facilitar a extensão de aplicativos e sistemas, permitindo que qualquer pessoa possa alterar o comportamento de uma parte do programa sem a necessidade de re-compilação e usando uma linguagem de alto-nível de abstração [Valente 2005].

Desde os primeiros sistemas operacionais, os scripts têm sido um grande aliado dos usuários de computadores como uma forma rápida e segura de agilizar tarefas muito repetitivas, aumentando, assim a produtividade do usuário. Um script é um arquivo com uma sequência de comandos que o usuário, eventualmente, esteja acostumado a repetir seguidamente. Com estes arquivos, o usuário pode

substituir grandes seções de trabalho por comandos simples que, por sua vez, disparam uma série de outros comandos [Nunes e Pereira 1999].

Já é tido como fato por algumas décadas que a melhor estratégia para aumentar a produtividade no desenvolvimento de software é o reaproveitamento de código, que por sua vez é uma consequência da qualidade da especificação dos processos. Entretanto, devido a falhas de engenharia de software, as especificações dos sistemas não costumam evitar a necessidade de grandes esforços na normalização de dados e na digitação de largos trechos de programa repetidas vezes. As linguagens de script por sua vez utilizam seus tipos auto-adaptáveis para simplificar a normalização de dados encorajando a reutilização de código fonte.

Os scripts podem ser executados em uma máquina virtual ou em um interpretador embarcado em uma aplicação. Quando executado em uma máquina virtual, o script está limitado ao conjunto de instruções nela programadas. Quando executado por meio de um interpretador embarcado, torna-se possível a criação de novos conjuntos de instruções para serem utilizados no script [Valente 2005].

2.2 Agentes

Segundo [Wooldridge 1999], “Um agente é um sistema e computador que está situado em algum ambiente e que é capaz de executar ações autônomas de forma flexível neste ambiente, a fim de satisfazer seus objetivos de projeto”. Sendo assim, um agente raciocina sobre o ambiente, sobre os outros agentes e decide racionalmente quais objetivos deve perseguir e quais ações deve tomar.

Não existe ainda um consenso quanto a definição de agente, mas podem ser caracterizados através de capacidades ou funcionalidades que possuam. Também não existe uma classificação universal para os agentes, entretanto a maioria aceita a autonomia como característica principal que distingue um software tradicional de um agente. Autonomia é identificada pela capacidade de um software em exercer controle sobre suas próprias ações [Ferreira e Bercht 2000].

Ainda de acordo com [Ferreira e Bercht 2000], uma interessante aplicação para o uso de agentes é a educação e treinamento. Os agentes autônomos projetados e desenvolvidos para dar apoio ao aprendizado humano, interagindo com educadores e educandos, de tal forma a facilitar o seu aprendizado, são chamados Agentes Pedagógicos.

Um agente não precisa incorporar todos os atributos citados. Até porque as características de um agente são dependentes do tipo de aplicação a que ele se propõe [Costa 1999].

A análise dos atributos que estão presentes nos agentes tem sido utilizada pelos pesquisadores para organizar os agentes em tipologias. Uma tipologia é uma classificação por tipos de agentes que possuem atributos em comum.

Em um RPG, existem vários tipos de NPCs. Existem aqueles que fazem parte de *quests* primárias,

secundárias e os que apenas preenchem cenário. O comportamento dos NPCs depende de vários fatores, como por exemplo, o contexto do cenário, os objetos que o jogador está portando, o diálogo que o jogador mantém com este agente e/ou outros agentes, o desempenho do jogador em uma ou mais atividades pedagógicas.

3. Resultados

No contexto do RPGEDU, todas as atividades foram feitas em pelo menos três graus de dificuldade, muitas delas com quatro graus de dificuldade. Um agente especial é responsável por monitorar o desempenho do jogador e influenciar o nível de dificuldade da atividade no momento de executar a atividade. Entretanto, apesar do comportamento deste agente estar implementado e funcionando corretamente, faltou uma árvore de decisão feita pelos pedagogos que informasse quando seria possível aumentar a dificuldade e quando poderia diminuir.

Antes do usuário iniciar uma atividade, o sistema verifica se a atividade está disponível ou se já não foi realizada, podendo então executá-la. As atividades pedagógicas são desenvolvidas utilizando alguns componentes de formulários – fornecidos pela biblioteca CEGUI – como *radio buttons*, *buttons* e *static images* – como mostrado na Figura 1. Dentro da atividade, quando o usuário pressiona um botão do mouse, o núcleo do sistema verifica se o usuário clicou em cima de algum componente e comunica ao script o nome do componente, para que este determine a ação que deve ser tomada.



Figura 1: Atividade mostrada através de script

Por todas as vantagens apresentadas sobre as linguagens de script, percebe-se que estas são boas opções para o desenvolvimento das atividades pedagógicas.

Para implementação dos agentes, dentre as tipologias disponíveis, este trabalho utiliza a de agentes reativos.

Agente reativo é o agente que tem a capacidade de detectar alterações ocorridas no ambiente e atuar em resposta a estas alterações [Machado et al 2004].

Cada agente possui uma regra para decidir qual ação deve ser executada ao interagir com o jogador. Ao final do artigo são encontrados os 6 principais diagramas de decisão dos agentes do RPGEDU.

Após estarem definidos os comportamentos dos agentes, estes foram implementados utilizando a linguagem Lua, onde máquinas de estados finitos foram utilizadas para representar todos os possíveis comportamentos dos agentes. Várias ações podem mudar o estado de um agente. O próprio agente pode mudar seu estado enquanto conversa com o jogador e o resultado de atividades pedagógicas também são fatores que podem mudar o comportamento de agentes no contexto do jogo.

No contexto do RPGEDU, alguns agentes fornecem dicas para ajudar o jogador, caso eles detectem que o jogador não concluiu ainda ou está cometendo muitos erros em determinada atividade pedagógica. Os agentes podem monitorar a quantidade de erros e acertos do jogador ao realizar as atividades pedagógicas. O RPGEDU recolhe informações do aluno através dos agentes e das atividades pedagógicas. Com esses dados é possível fazer uma análise para descobrir o ponto fraco e o ponto forte do aluno em cada matéria estudada.

Sabendo das dificuldades o agente se encarrega de tomar as decisões que o professor tomaria para reforçar o aluno na matéria que tem dificuldades. Assim, de acordo com a evolução do aluno, o agente seleciona atividades com grau de dificuldade mais elevado para os alunos mais avançados. Caso o aluno sinta dificuldade em solucionar alguma atividade o agente muda a forma de abordagem para que o aluno entenda a questão.

3. Discussões e conclusões

Com esse projeto pôde-se criar um jogo de RPG com uma estrutura flexível baseada em scripts e arquivos XML para configuração, que permite um conteúdo flexível e facilmente extensível. É importante que as aplicações desenvolvidas, sejam elas, jogos ou outros sistemas, sejam flexíveis, não somente para que lhe sejam adicionadas novas características, mas também para facilitar o seu processo de manutenção.

Uma vez detectado algum problema nessas aplicações, é possível ir direto ao ponto aonde o problema ocorre e aplicar as correções sem alterar a aplicação principal, dessa forma, não é necessário realizar a re-compilação de todo o sistema, apenas do módulo (ou arquivo de script) que está tendo problemas.

A utilização de linguagens de script não tem benefícios somente no produto final, mas também no seu processo de desenvolvimento, pois além de eliminar a necessidade de re-compilar o aplicativo a qualquer alteração – o que resulta em maior agilidade no desenvolvimento –, torna a aplicação mais modular. Outro aspecto importante a ser observado é que as aplicações escritas utilizando linguagens de script podem ser facilmente alteradas, inclusive por pessoas

que não possuam um conhecimento muito aprofundando em programação ou mesmo do sistema em que se está fazendo a manutenção.

Arquivos XML foram utilizados nesse projeto para definir os conteúdos a serem aplicados. O uso de arquivos XML é mais uma opção para tornar o conteúdo da aplicação flexível, e parametrizável. Permitindo que menos definições de atributos fiquem fixas no código principal da aplicação, ajudando a dar maior dinamismo e flexibilidade. O uso de arquivos de linguagens de script e de arquivos XML para parametrização de conteúdo é benéfico tanto para jogos como para os demais tipos de aplicativos.

Após todos os estudos, pesquisas e aplicações realizadas, conclui-se que a Ogre3D pode ser uma opção ótima para desenvolver aplicações multimídia para um público-alvo disposto a adquirir tecnologias mais modernas em hardware dedicado a renderização de imagens. E pode ser inviável na falta de bons recursos de hardware.

O uso de linguagens de script mostrou-se eficiente no desenvolvimento das atividades pedagógicas e trouxe uma maior agilidade no desenvolvimento por sua flexibilidade, praticidade e simplicidade. Uma das maiores vantagens da utilização de linguagem de script é a possibilidade de fazer alterações (adaptações, expansões, etc) nos jogos sem precisar recompilar todo o projeto, pois para tal, basta fazer os ajustes nos arquivos dos scripts que o jogo já estará com as modificações efetuadas.

Referências

- COSTA, M. T. C. 1999. "UMA ARQUITETURA BASEADA EM AGENTES PARA SUPORTE AO ENSINO A DISTANCIA". DISPONÍVEL EM [HTTP://WWW.EPS.UFSC.BR/TESES99/THIRY/](http://www.eps.ufsc.br/teses99/thiry/) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- FERREIRA, L. F. E BERCHT, M. 2000. "AGENTES PEDAGÓGICOS COMO APOIO À AVALIAÇÃO DE COMPETÊNCIA TÉCNICA EM EDUCAÇÃO E PRÁTICA MÉDICA". DISPONÍVEL EM [HTTP://WWW.C5.CL/IEINVESTIGA/ACTAS/RIBIE2000/PAPER/187/INDEX.HTM](http://www.c5.cl/ieinvestiga/actas/RIBIE2000/PAPER/187/INDEX.HTM) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- FRANCO, M. A. 2000. "INFORMÁTICA E PODER: UMA LEITURA DE FOUCAULT". DISPONÍVEL EM [HTTP://WWW.REVISTA.UNICAMP.BR/INFOTEC/EDUCACAO/EDUCACAO.HTML](http://www.revista.unicamp.br/infotec/educacao/educacao.html). [ACESSADO EM 12 DE DEZEMBRO DE 2005].
- FURTADO, A. W. B., SANTOS, A. L. M., GOMES, A. S. 2003. DISPONÍVEL EM [WWW.CIN.UFPE.BR/~ALMS/PDF/](http://www.cin.ufpe.br/~alms/pdf/) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- KUMICK, C. 2003. "O RPG NA EDUCAÇÃO". DISPONÍVEL EM [HTTP://WWW.HISTORIAS.INTERATIVAS.NOM.BR/EDUC/APLICAR.HTM](http://www.historias.interativas.nom.br/educ/aplicar.htm) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- MACHADO, M. L. M., SOUZA, D. G., SOUZA, J. A., DANDOLINI, G. A., SILVEIRA, R. A., DANDOLINI, G. A. 2004. "RPG: UMA ABORDAGEM EMPREGANDO SISTEMAS MULTIAGENTES". RENOTE REVISTA NOVAS TECNOLOGIAS NA EDUCAÇÃO, PORTO ALEGRE.
- NUNES, M. P., PEREIRA, T. P. 1999. "USO DE LINGUAGENS DE SCRIPT COMO FERRAMENTAS DE APOIO AO ENSINO". DISPONÍVEL EM [HTTP://DMF.FURG.BR/~PYTHON/SCRIEDU.HTML](http://dmf.furg.br/~python/scriedu.html) [ACESSADO EM 22 DE DEZEMBRO DE 2007].
- TANAKA, M. 2005. "RPG E EDUCAÇÃO". DISPONÍVEL EM [HTTP://WWW.AOMESTRE.COM.BR/CELL/ARQUIVO/V25.HTM](http://www.aomestre.com.br/cell/arquivo/v25.htm) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- VALENTE, J. A. 2002. "DIFERENTES USOS DO COMPUTADOR NA EDUCAÇÃO". DISPONÍVEL EM: [HTTP://UPF.TC.BR/~CAROLINA/POS/VALENTE.HTML](http://upf.tc.br/~carolina/pos/valente.html) [ACESSADO EM 12 DE DEZEMBRO DE 2007].
- WOOLDRIDGE, M. 1999. "Multiagent Systems". A Modern Approach to Distributed Artificial Intelligence. MIT Press.

Desenvolvimento de Jogos para o Aperfeiçoamento na Aprendizagem de Disciplinas de Ciência da Computação

Luciano A. Digiampietri
Universidade de São Paulo (EACH-USP)
digiampietri@usp.br

Diogo D. Kropiwiec
Universidade Estadual de Campinas (IC-UNICAMP)
diogo@las.ic.unicamp.br

Abstract

Atualmente, os professores de ensino superior em cursos ligados à computação apresentam duas grandes preocupações. A primeira é quanto à diminuição de candidatos por vaga nesses cursos. O que, de certa forma, contradiz as necessidades do mercado de trabalho que ainda carece de pessoal qualificado nessa área. A segunda preocupação diz respeito à insatisfação dos graduandos com algumas disciplinas, principalmente por não conseguirem vislumbrar a aplicação prática de diversos conceitos teóricos, o que pode, em casos extremos, aumentar a evasão destes cursos.

O objetivo deste projeto é ajudar na resolução destes problemas por meio do desenvolvimento de um ambiente computacional para a implementação e o gerenciamento de jogos de computador que poderá ser usado por professores de diversas disciplinas como exemplo prático dos diversos conceitos teóricos ensinados na sala de aula. Além disso, grande parte do conteúdo desenvolvido, incluindo código fonte, jogos, tutoriais, simuladores e apresentações, estará disponíveis via Internet, possibilitando que qualquer um interaja com os mesmos.

Keywords:: Inteligência Artificial; Desenvolvimento de Jogos; Jogos na Educação; Ensino de Computação

1 Introdução

Atualmente, existem dois grandes desafios no ensino de cursos de graduação ligados à Ciência da Computação: tornar o curso mais atrativo aos estudantes do ensino médio para que estes ingressem nesses cursos e tornar o curso mais interessante para aqueles que já o estão cursando [Guzdial 2003; Forte and Guzdial 2005].

Após a grande procura por cursos ligados à computação, que ocorreu principalmente na década de 1990, observamos uma constante queda na procura por este tipo de curso. O número de candidatos por vaga nas principais instituições públicas de ensino no país está caindo em todos os níveis de ensino (graduação, mestrado e doutorado). Há diversas hipóteses sobre esse fenômeno, tais como (i) a computação não é mais o curso da “moda”; (ii) a grande maioria dos alunos do ensino médio já sabe “utilizar” o computador e por isso não vêem necessidade de um curso de graduação neste assunto; ou mesmo (iii) a impressão de que o perfil de uma pessoa formada nestes cursos é de alguém que ficará o dia inteiro sentado em frente ao computador digitando coisas (potencialmente desinteressantes) sem se envolver com desafios de outras áreas (consideradas mais interessantes por muitos alunos).

Por outro lado, considerando os graduandos em cursos de computação (Ciência da Computação, Sistemas de Informação, Análise de Sistemas, etc), há duas reclamações constantes: (i) as disciplinas desses cursos são ministradas de maneira disjuntas (sem existir uma grande ligação entre o conteúdo de uma disciplina em relação às disciplinas já ministradas) e (ii) há disciplinas excessivamente teóricas, nas quais os alunos não conseguem e/ou não tem oportunidade de aplicar o aprendizado teórico em problemas reais.

Acreditamos que a crescente desmotivação pela procura por cursos de computação dá-se devido à desinformação quanto ao conteúdo do curso; quanto às possibilidades multidisciplinares permitidas pela área; e, até mesmo, sobre o mercado de trabalho. Este fato foi comprovado muitas vezes no evento Unicamp de Portas Abertas (UPA), no qual a universidade abre suas portas e apresenta alguns de seus projetos a estudantes do ensino médio. Nos estandes dedicados aos cursos de computação, é constante a surpresa desses

alunos ao saber que todos os projetos apresentados (envolvendo os mais variados assuntos) são projetos em computação.

Já as críticas dos graduandos em cursos de computação estão intimamente ligadas a falta de um aspecto prático das disciplinas. Aspecto este que é muito procurado pelo mercado de trabalho e que, muitas vezes, não é satisfeito pelos cursos das universidades públicas. De um modo geral, os cursos de computação nas universidades públicas são mais abrangentes e mais teóricos do que os cursos equivalentes de universidades particulares; o que é muito bom, considerando os diferentes objetivos destes dois tipos de instituição. Porém, julgamos que um projeto que possibilite a aplicação prática dos diversos conceitos teóricos pode ajudar o aluno a assimilar o conteúdo teórico, além de servir como um facilitador na entrada deste aluno no mercado de trabalho.

Este artigo apresenta um ambiente de desenvolvimento de jogos para ajudar a enfrentar estes dois desafios. Enquanto o principal enfoque será no segundo desafio, por meio do desenvolvimento de atividades práticas de Algoritmos e Estruturas de Dados, Inteligência Artificial, Interface Humano-Computador e Engenharia de Software, o primeiro desafio será enfrentado como consequência do segundo, por meio da disponibilização dos softwares desenvolvidos, de materiais de apoio e de palestras a serem apresentadas em feiras de profissões e, possivelmente, nas escolas de ensino médio.

O restante deste artigo está organizado da seguinte maneira. Seção 2 detalha as metas e objetivos. A Seção 3 sumariza alguns trabalhos correlatos. Seção 4 apresenta a infra-estrutura atual. A Seção 5 apresenta o trabalho que está sendo atualmente desenvolvido e os próximos passos. Por fim, a Seção 6 apresenta as conclusões e um resumo das contribuições.

2 Objetivos

Este artigo apresenta o protótipo de um ambiente computacional a ser utilizado por professores e alunos para o aperfeiçoamento no aprendizado de disciplinas ligadas à computação. O domínio “desenvolvimento jogos” foi escolhido pelos seguintes motivos: (i) por envolver os mais diversos assuntos da computação; (ii) por ser de interesse dos alunos e das empresas; (iii) por apresentar desafios científicos; e (iv) pela sua grande visibilidade para alunos do ensino médio. A seguir, cada um desses motivos é descrito.

(i) Envolvimento dos diversos assuntos da computação.

O desenvolvimento de jogos envolve aspectos de Engenharia de Software: projeto, implementação e testes; Interface Humano-Computador: interface gráfica, aspectos de acessibilidade e jogabilidade; Inteligência Artificial: representação de conhecimento; algoritmos de busca para implementação de *bots*; Sistemas Distribuídos; Redes Sociais; Algoritmos e Estruturas de Dados; entre outros. Cada uma destas disciplinas poderia tirar proveito deste ambiente para o desenvolvimento de projetos práticos.

(ii) Interessante aos alunos e às empresas.

Diversos alunos têm interesse no desenvolvimento de jogos não apenas pelo fato de jogos serem divertidos, mas também porque o resultado do desenvolvimento é “palpável” no sentido que, assim que o protótipo de um jogo estiver desenvolvido, o desenvolvedor chama usuários para testá-lo e imediatamente receberá um retorno se o jogo está agradando ou não. Consideramos que esta experiência com usuários reais é muito importante, porém inexistente para muitos graduandos. Por outro lado, o mercado de jogos de computadores tem crescido bastante nos últimos anos e há uma demanda não atendida por profissionais com alguma experiência na

área. Desta forma, alunos que participassem ativamente no desenvolvimento do ambiente proposto provavelmente teriam vantagens ao tentar ingressar nas empresas desta área.

(iii) Desafios científicos. Todas as disciplinas de computação apresentadas no item (i) apresentam desafios científicos (por exemplo, questões de acessibilidade, algoritmos de inteligência artificial, sistemas distribuídos, etc). A principal meta científica deste projeto é ter alunos de iniciação científica trabalhando nesses desafios enquanto ajudam a desenvolver e ampliar o ambiente computacional proposto.

(iv) Visibilidade aos alunos do ensino médio. Alunos do ensino médio (assim como muitos outros adolescentes e jovens) têm muito interesse em jogos de computadores. Geralmente, jogos são vistos apenas como uma atividade lúdica, porém eles têm grande utilidade como ferramentas educativas. Pretende-se que o ambiente proposto forneça ferramentas (jogos e outros materiais didáticos) para ajudar no ensino de disciplinas tais como lógica, algoritmos, física, etc. O ambiente computacional proposto também fornecerá interfaces simples para a criação de novos jogos ou fases sem a necessidade de conhecimento de programação. Por exemplo, a criação de uma nova fase do jogo Newmings (ver Seção 4.e), ou a especificação de um jogo de xadrez onde todas as peças, exceto o rei, sejam cavalos. Este tipo de atividade permite que usuários sem conhecimentos em computação interajam na construção do ambiente. Julgamos que esta combinação entre este viés colaborativo e o viés competitivo comumente encontrado nos jogos pode ser muito construtiva. Além disso, pretendemos executar tarefas (palestras, tarde de jogos, etc) juntamente com a organização Atlética de Alunos da EACH-USP a fim de divulgar os mais variados aspectos dos cursos de computação e tentando aumentar a visibilidade destes para os alunos do ensino médio.

3 Trabalhos Correlatos

Diversos experimentos foram realizados tanto na área de Educação em Informática quanto em Jogos de Computadores para demonstrar que o uso de jogos em disciplinas introdutórias de cursos ligados à computação é benéfico para motivar os alunos e aumentar o aprendizado dos mesmos [Forte and Guzdial 2005; Clua 2008].

Jogos têm sido usados em diversas iniciativas como incentivo ao aprendizado para crianças, adolescentes, jovens e adultos [Klawe 1999; Distasio and Way 2007]. Inclusive, novas metodologias de ensino surgiram, como o Aprendizado Baseado em Jogos que está sendo utilizada em diversos sistemas e, atualmente, existem eventos científicos inteiramente dedicados a esta metodologia [Tan et al. 2007; Burgos et al. 2008].

Enquanto a maioria dos trabalhos existentes está focada no uso de jogos para o aprendizado de disciplinas introdutórias em computação, nosso trabalho se destaca em três características: (i) o ambiente proposto visa ajudar no aprendizado de disciplinas desde as introdutórias como Algoritmos e Estruturas de Dados, até disciplinas mais específicas, tais como, Inteligência Artificial, Engenharia de Software, Interface Humano-Computador e Tecnologias para Web; (ii) participação ativa dos alunos na especificação e desenvolvimento do ambiente: além dos módulos básicos especificados e desenvolvidos pelos proponentes, o restante do sistema está sendo implementado de acordo com as sugestões dos alunos; (iii) ambiente de pesquisa em computação: um dos objetivos do ambiente é apoiar atividades científicas nas mais diversas áreas, como Inteligência Artificial, Segurança de Redes e Interfaces.

4 Infra-estrutura atual

O ambiente proposto está em desenvolvimento e já possui os seguintes sistemas (em fase de teses): (a) servidor de jogos; (b) aplicação deflexion¹; (c) aplicação multi-jogadora; (d) bots jogadores de jogo da velha; (e) Newmings; (f) gerador de Sudoku;

¹Deflexion é um jogo de estratégia que utiliza espelhos e laser: www.deflexion.biz

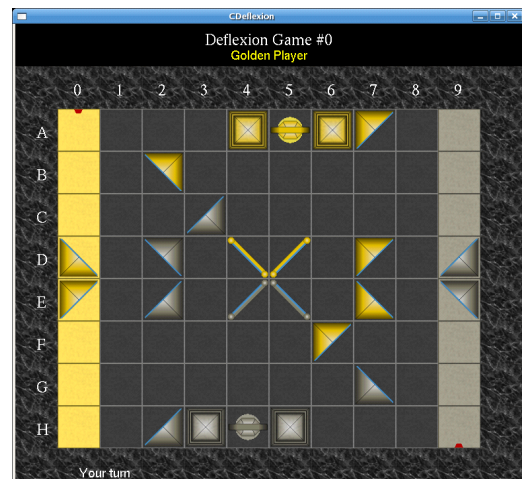


Figure 1: Cópia de tela da Aplicação Deflexion

e (g) conjunto de jogos em Prolog. A idéia do ambiente é possuir uma variedade de jogos desenvolvidos utilizando tecnologias distintas para, assim, poderem ser utilizados nas atividades práticas de diversas disciplinas. A seguir, cada um destes sistemas/programas é descrito:

a) Servidor de Jogos: trata-se de um servidor de jogos de tabuleiro on-line. Já possui implementada a lógica dos seguintes jogos: jogo da velha 2D (convencional) e 3D; damas; xadrez; e deflexion. O servidor está implementado em Java (sendo independente de plataforma) e as aplicações se comunicam com ele via mensagens SOAP, o que permite que aplicações desenvolvidas nas mais diversas linguagens de programação e sistemas operacionais possam interagir com o servidor. Este sistema é composto por três módulos principais: Jogador, Gerenciador de Jogos e Interface de Comunicação. O Jogador armazena as informações de cada jogador (cadastro) além de um histórico de seus resultados. O módulo Gerenciador de Jogos contém as classes básicas para a criação de um jogo: Tabuleiro, Peça e Jogo. Para se implementar um novo jogo basta estender cada uma dessas classes. Dentro do Gerenciador de Jogos também estão as extensões dessas classes para os jogos já implementados (jogo da velha, damas, xadrez, etc). O módulo Comunicação é responsável por manter portas de comunicação para o recebimento das mensagens SOAP, tradução dessas mensagens para o módulo Gerenciador de Jogos e envio das respostas às mensagens recebidas. Uma outra característica interessante deste servidor é que há duas maneiras de se adicionar bots: (i) estendendo as classes já implementadas para o processamento automático de jogadas ou (ii) através de um serviço Web que receba o estado do jogo e sugira a próxima jogada.

b) Aplicação Deflexion: é um “programa cliente” que interage com o Servidor de Jogos para se jogar o jogo deflexion. Enquanto toda a lógica do jogo encontra-se no servidor, esta aplicação é responsável por fornecer uma interface gráfica para que um usuário possa facilmente interagir com o servidor. Esta aplicação foi desenvolvida em C++ e utiliza algumas bibliotecas C++ exclusivas do Linux. Por se tratar de uma aplicação local, o usuário deve executá-la de sua máquina e passar como parâmetro o endereço do servidor de jogos (endereço IP e número da porta de comunicação). A Figura 1 apresenta uma cópia de tela da Aplicação Deflexion.

c) Aplicação Multi-Jogadora: esta aplicação permite que um usuário jogue, via Internet, qualquer um dos jogos do servidor. Enquanto a Aplicação Deflexion foi desenvolvida exclusivamente para interagir com o jogo deflexion (e, por isso, possui uma interface gráfica muito mais detalhada e específica), a Aplicação Multi-Jogadora possui uma interface genérica sem grandes detalhes gráficos, mas que possibilita a interação com qualquer um dos jogos. Desenvolvida inicialmente para testar cada um dos novos jogos incluídos no Servidor de Jogos, ela também pode ser utilizada



Figure 2: Cópia de tela do programa Newmings

por aqueles que queiram interagir com o servidor de jogos sem a necessidade de baixar nenhuma aplicação. Esta aplicação foi desenvolvida em Perl e utiliza o formato CGI para possibilitar a interação com o usuário via navegador de Internet.

d) Bots Jogadores de Jogo da Velha: este sistema não está diretamente relacionado com os anteriores, pois sua intenção é apresentar uma aplicação simples de aspectos teóricos; neste caso, sobre Inteligência Artificial. Além de permitir ao usuário jogar Jogo da Velha, este sistema também fornece três *bots* (do inglês, *robots*, ou seja, programas de computador capazes de jogar um dado jogo) que implementam diferentes métodos da Inteligência Artificial: (i) busca MiniMax; (ii) sistema especialista (baseado em regras); e (iii) uso de dicionário de jogadas. O Jogo da Velha foi escolhido para este sistema por se tratar de um jogo extremamente simples, no qual é fácil explicar e exemplificar o funcionamento de cada um dos métodos de Inteligência Artificial implementados.

e) Newmings: inspirado no jogo Lemmings de 1991, este programa apresenta um jogo de lógica cujo objetivo é salvar um determinado número de pingüins. Ao usuário, cabe dar algumas capacidades específicas aos pingüins de forma a atingirem seu objetivo. Este jogo foi desenvolvido em C++ e uma das características interessantes é que, para se adicionar novas fases/níveis basta a especificação de dois arquivos. Ou seja, um usuário que não conhece nada de computação pode “criar um novo jogo” sem precisar programar uma linha de código. A Figura 2 apresenta uma cópia de tela da primeira fase do programa Newmings.

f) Gerador de Sudoku: gera automaticamente “tabuleiros” de Sudoku. Desenvolvido em C# (.net), o usuário pode configurar o grau de dificuldade solicitando ao programa que varie a quantidade de números que precisarão ser preenchidos para se resolver o Sudoku. A Figura 3 apresenta uma cópia de tela do Gerador de Sudoku.

g) Jogos em Prolog: este sistema é composto por quatro jogos desenvolvidos em Prolog: (i) sistema especialista jogador de jogo da velha; (ii) sistema especialista jogador de truco; (iii) jogador de damas, baseado em busca MiniMax; e (iv) o Mundo de Wumpus (nome original: *Hunt the Wumpus*).

Por enquanto, estes sistemas estão disponíveis para testes apenas para os alunos de algumas disciplinas específicas dentro da Universidade de São Paulo, por exemplo, Introdução a Ciência da Computação, Algoritmos e Estruturas de Dados e Inteligência Artificial. Esta interação com os alunos está possibilitando o aperfeiçoamento do ambiente proposto para que, nos próximos meses, este seja disponibilizado via Internet.

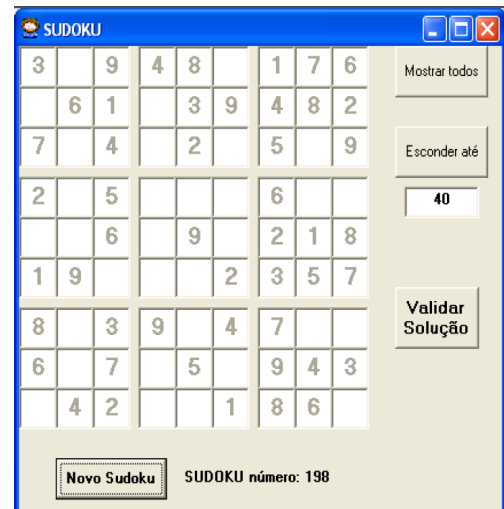


Figure 3: Cópia de tela do programa Gerador de Sudoku

5 Próximos passos

Conforme apresentado na seção anterior, diversos sistemas já foram desenvolvidos (ou prototipados). A tarefa atual consiste em melhorar a documentação do que já foi implementado, desenvolver novos programas, estender e integrar esses sistemas e disponibilizar tudo via Internet criando um ambiente interativo onde qualquer usuário possa jogar, criar novos jogos e baixar jogos e materiais didáticos relacionados.

Neste semestre, parte do sistema está sendo testado junto aos alunos das disciplinas Inteligência Artificial e Introdução a Ciência da Computação II. Para o semestre que vem, serão desenvolvidos simuladores para apoiar o ensino de algoritmos clássicos de estruturas de dados, tais como, gerenciamento de filas, pilhas, listas e árvores, Torres de Hanói, algoritmos para a resolução do Cubo Mágico, etc.

Também estão sendo desenvolvidas ferramentas para permitir um torneio de *bots* para todos os jogos que compõem o Servidor de Jogos citado anteriormente.

Futuramente, pretendemos desenvolver um projeto em Aprendizado Baseado em Jogos que será acrescentado ao ambiente proposto. No Aprendizado Baseado em Jogos, jogos são criados com a única finalidade de serem ferramentas para o aprendizado. Esta área tem se desenvolvido bastante atualmente, tanto em aspectos teóricos quanto em aspectos práticos. Acreditamos que este tipo de projeto combinado ao ambiente apresentado neste artigo serão uma grande contribuição para o aprendizado em computação.

O conteúdo desenvolvido até o momento está sendo organizado e documentado e será disponibilizado via Internet em dois locais distintos: (i) no site do projeto (<http://www.uspleste.usp.br/digiampietri/jogos>) e no Portal *Microsoft Faculty Conexión* (<http://www.microsoft.com/education/facultyconnection/br>).

6 Conclusões

Este artigo apresentou um ambiente computacional para o desenvolvimento e disponibilização de jogos de computador que visa obter contribuições em três diferentes âmbitos: ensino, pesquisa e extensão.

Quanto ao ensino, as contribuições esperadas são a disponibilização de um ambiente computacional para a elaboração de atividades práticas ligadas às áreas de (i) Inteligência Artificial por exemplo, agentes de software, sistemas especialistas, algoritmos de busca e representação de conhecimento; (ii) Engenharia de Software: projeto de software, testes, programação extrema (XP) e padrões de projeto; (iii) Algoritmos e Estruturas de Dados; (iv) Sistemas Distribuídos: sistemas interoperáveis e padrões para a Web;

e (v) Interface Humano-Computado: interfaces gráficas interativas. Além deste ambiente computacional para auxílio no aprendizado de disciplinas em computação que poderá auxiliar centenas de alunos por semestre em nossa unidade de ensino (além dos usuários esporádicos ou mesmo possíveis colaborações com outras unidades de ensino), acreditamos que a interação entre os alunos de computação e este ambiente poderá representar um diferencial no currículo destes alunos, pois muitas empresas de software procuram profissionais que já tenham experiência em uma ou mais das seguintes áreas: programação ágil, desenvolvimento de jogos, desenvolvimento de sistemas interoperáveis e uso de padrões de projeto.

No âmbito de pesquisa, este projeto apresenta contribuições potenciais em dois níveis diferentes. O primeiro está relacionado ao uso do ambiente: pelo fato do ambiente computacional proposto ser interoperável, ele poderá ser usado para testar comparativamente diferentes soluções para, por exemplo, algoritmos jogadores de um dado jogo (*bots*); ou mesmo teste de interfaces, etc. Ou seja, o ambiente proposto pode servir de apoio a diversas atividades de pesquisa. A segunda contribuição é mais direta e está relacionada ao desenvolvimento do ambiente propriamente dito. Existem diversos desafios científicos quanto à interação humano-computador, incluindo questões de acessibilidade e usabilidade; algoritmos de inteligência artificial; e processo de especificação, desenvolvimento e teste de software. Pretendemos comparar e estender as soluções propostas para cada tipo de problema e, com isso, gerar resultados inovadores em algumas destas áreas.

As contribuições ligadas à extensão universitária se referem ao impacto positivo que a disponibilização do ambiente proposto, incluindo material didático, jogos de lógica e código fonte, pode trazer aos alunos do ensino médio. Além de difundir os cursos ligados à computação apresentando aspectos de como estes cursos podem envolver atividades multidisciplinares, o material de apoio também poderá ser útil para aperfeiçoar o entendimento de lógica além de servir como uma introdução destes alunos à área de programação de computadores.

References

- BURGOS, D., MORENO-GER, P., SIERRA, J. L., FERNÁNDEZ-MANJÓN, B., SPECHT, M., AND KOPER, R. 2008. Building adaptive game-based learning resources: The integration of ims learning design and. *Simul. Gaming* 39, 3, 414–431.
- CLUA, E. W. G. 2008. A game oriented approach for teaching computer science. In *Anais do XXVIII Congresso da SBC, Workshop sobre Educação em Computação (WEI)*, 10–19.
- DISTASIO, J., AND WAY, T. 2007. Inclusive computer science education using a ready-made computer game framework. *SIGCSE Bull.* 39, 3, 116–120.
- FORTE, AND GUZDIAL. 2005. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE TE: IEEE Transactions on Education* 48.
- GUZDIAL, M. 2003. A media computation course for non-majors. *SIGCSE Bull.* 35, 3, 104–108.
- KLAWE, M. M. 1999. Computer games, education and interfaces: the e-gems project. In *Proceedings of the 1999 conference on Graphics interface '99*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 36–39.
- TAN, P.-H., LING, S.-W., AND TING, C.-Y. 2007. Adaptive digital game-based learning framework. In *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, ACM, New York, NY, USA, 142–146.

Simulação de TV Holográfica Utilizando o Wiimote

André Luiz J. D. Tschaffon, Breno M. de Paula, Rafael C. Slobodian, Esteban Clua, Anselmo Montenegro.

Universidade Federal Fluminense, Instituto de Computação, Brasil

Resumo

Em tempos em que interação homem-máquina vem evoluindo, utilizando tecnologias novas, como telas sensíveis a toque e receptores de áudio, o *wiimote*, da Nintendo Wii, chega para somar a esses novos periféricos, sendo uma tecnologia relativamente barata levando-se em conta sua complexidade, possibilitando uma grande difusão e aceitação no mercado.

Neste artigo, será desenvolvida uma aplicação que utiliza esta tecnologia para criar uma solução capaz de simular o efeito de paralaxe produzido em visão estéreo, sem no entanto utilizar monitores ou óculos especiais. Destaca-se que este trabalho consiste em uma implementação alternativa àquela desenvolvida em [1].

Keywords: wiimote, TV holográfica, visão estéreo

1. Introdução

Com a chegada do Nintendo Wii ao mercado, uma revolução em quesito de jogabilidade foi criada, possibilitando novas experiências de usabilidade. O *wiimote*, *joystick* inovador do console, torna a maneira de jogar muito mais natural e interessante para pessoas que sentem dificuldade de se divertir com os controles tradicionais.

O *wiimote* possui tecnologias que servem para captar movimentos e posições do controle no espaço. Quando acoplado a um computador, uma nova forma de interação pode ser criada, substituindo o tradicional mouse por um sensor capaz de capturar posições de mundo e movimentações.

Este projeto faz uso da captura da posição da câmera infravermelha para criação de um novo tipo de posicionamento virtual de câmera em mundos tridimensionais.

Ao posicionar dois LEDs infravermelhos nas extremidades de uma barra, chamada de **barra sensora**, pode-se utilizar a câmera infravermelha embutida no *wiimote* para calcular, de forma aproximada, a posição em que o observador se encontra. Esta barra não necessita ser a mesma encontrada no Wii. Através destas informações obtidas, reposicionamos a câmera em um ambiente tridimensional, sem modificar o plano de projeção, transformando o dispositivo de saída (Monitor ou TV) em uma espécie de janela ou moldura de quadro sem conteúdo, pela qual se pode observar uma cena tridimensional. O resultado visual para o observador é bastante impressionante, pois à medida que este se movimenta, tem-se a impressão clara e real de

profundidade do ambiente, chegando inclusive a propiciar uma impressão de holografia. Como a tecnologia envolvida na TV holográfica é muito cara, tornando assim o preço deste produto extremamente alto, esta técnica de gerar uma falsa holografia através do *wiimote* é menos custosa e muito bem recebida pelo usuário.

Como a câmera infravermelha é um dispositivo que não captura profundidade e apenas provê informações de posição em X e Y, foi criada uma estrutura para cálculo da movimentação no eixo Z, de modo que as posições no mundo real fossem proporcionais no mundo virtual. O objetivo de criar um mundo com visão acurada foi uma das metas deste projeto, diferenciando-se da proposta de [1].

A implementação do sistema proposto foi feita em XNA, utilizando o Microsoft Visual Studio 2005 e consiste em um cenário tridimensional que simula um panorama (uma foto de um determinado ambiente), a partir de uma imagem gerada como textura interna de um cilindro, com a câmera posicionada no interior do mesmo.

2. Wii Remote

O *Wii Remote* (mais popularmente conhecido por *wiimote*) é um *joystick* com formato que lembra um controle remoto e possui basicamente 3 recursos: a detecção de movimentos através de acelerômetros de três eixos, a detecção de orientação espacial através de uma câmera infravermelha interna e uma porta de conexão para extensão à outros dispositivos.

O acelerômetro capta toda e qualquer aceleração feita em um dos eixos possíveis (X, Y e Z), além de detectar também movimentos de rotação sobre os eixos X (*pitch*), Z (*roll*) e Y (*yaw*).

A câmera infravermelha é semelhante a uma câmera de vídeo comum que, entretanto, capta apenas sinais com espectros infravermelhos. Possui uma resolução de 1024x768 *pixels* e ângulo de abertura de 45 graus. Possui também um processador interno que calcula o posicionamento de até quatro pontos infravermelhos simultaneamente e opera em uma frequência de 100 Hz.

A sua porta de conexão é uma interface que possibilita acoplar outros diferentes dispositivos de entrada, como o *Nunchuck*, dispositivo que possui uma alavanca analógica, dois gatilhos e um acelerômetro interno, e o *Classic Controller*, dispositivo que possui duas alavancas analógicas e se assemelha muito aos antigos *joysticks* de outro console da mesma empresa, o Super-Nintendo. Vale citar também que o *wiimote*

possui um pequeno auto-falante interno, utilizado para emitir sons referentes à utilização do mesmo em um determinado momento do jogo (para aumentar a imersão no jogo) e recurso de *rumble*, que emite uma vibração de acordo com a interação.

A conexão e a comunicação do *wiimote* com computadores podem ser feitas através de uma interface *Bluetooth*, facilmente configurada.



Figura 1: Todos os ângulos do *wiimote*

3. TV Holográfica

Holografia é o termo usado para registro ou apresentação de imagens em três dimensões. Foi definida, apenas na teoria, pelo húngaro Dennis Gabor[2]. Tem como fundamento que cada parte do holograma possui a informação do todo.

A partir desta idéia, a holografia na televisão se tornou um desejo de todos, pois seria uma total revolução na área tecnológica. O fato de não poder ser vista simultaneamente por mais de um observador é um de seus principais problemas.

O primeiro modelo de TV tridimensional sem acessórios para o observador foi criado em 1988[3]. Até a década de 90 foram criados vários protótipos de televisões holográficas, mas nenhum com efetivo sucesso.

A revolução nesta área de pesquisa surgiu em 2007, com o lançamento de uma TV holográfica de projeção para exposições em ambientes 3D, feita em vidro, com tela widescreen de 80 polegadas (aproximadamente dois metros), como podemos observar em [4].

Esta TV transmite ao usuário o efeito de ter a imagem pairando no ar. E, quando o projetor é desligado, a TV parece com um simples vidro.

O preço destes aparelhos e a dificuldade de se obter resultados convincentes com holografias são fatores fundamentais para a busca de novas tecnologias que sejam mais baratas, mas que ao mesmo tempo possibilitem ao usuário ter uma visualização mais realista de uma determinada imagem, criando assim uma falsa sensação de holografia.



Figura 2: Televisão holográfica de 80"

4. Trabalhos Relacionados

Esse artigo é baseado no trabalho de Jhonny Chung Lee [1], da universidade de Carnegie Mellon.

A proposição feita por Lee é que, considerando o fato do Nintendo Wii ser um videogame extremamente popular, o *wiimote* é um dos dispositivos de interface mais utilizados atualmente possuindo uma grande massa de usuários.

A grande visão do seu trabalho foi que, após conectar o *wiimote* a um PC, podemos torná-lo um detector de movimentos livres com os recursos disponíveis no controle, sendo o mais usado no projeto a captação de sinais infravermelhos posicionados convenientemente de acordo com cada aplicação.

Na utilização do console, o *joystick* fica na mão do jogador, e a barra com os LEDs infravermelhos fica posicionada abaixo ou acima do dispositivo de saída de imagem.

Na aplicação proposta, estas posições são invertidas, ou seja, a barra sensora, que corresponde aos dois LEDs é que fará os movimentos. O controle será utilizado para capturar as posições destes pontos de luz através da câmera infravermelha, e deve ficar no local em que numa configuração padrão deveria ficar a barra de LEDs.

Nesse formato de captura podem ser criados vários tipos de interfaces, tais como, pontos de luz nas pontas dos dedos.

A idéia principal do trabalho de Lee[1], que inspirou este projeto, foi calcular através das informações fornecidas pelo controle, a posição do observador, passando essas medidas para a câmera do ambiente virtual, alimentando a posição da mesma com as coordenadas da posição do observador. Sendo assim, ocorre uma simulação de falsa holografia, pois o monitor do computador passa a impressão de ser apenas uma janela, que possibilita a visualização de um espaço tridimensional de forma diferente das apresentadas nos jogos e aplicativos convencionais.

5. Implementação

Este projeto tem como objetivo aproximar-se da abordagem feita em [1], porém aplicando conceitos de componentização, de forma a permitir um reuso em outros projetos e jogos. Para implementar o protótipo, foi utilizada a linguagem C# em conjunto com o plataforma XNA Game Studio e a ferramenta Microsoft Visual Studio 2005.

Para a comunicação entre o computador e o *wiimote*, utilizou-se uma biblioteca desenvolvida em C# por Brian Peek [5], capaz de mapear todos os sinais enviados pelo *joystick* para o computador, via *Bluetooth*. O uso da biblioteca foi imprescindível, pois além de fazer a conexão entre os dispositivos, ela fornece uma interface única para o acesso aos seus dados, propiciando a utilização de todos os aparatos fornecidos pelo controle.

Os dados enviados pelo controle são repassados através de uma estrutura de relatórios. Estes são basicamente cadeias de bits pré-definidas que contêm as informações que se deseja obter ou fornecer ao dispositivo. O tipo do relatório é definido no momento da conexão, e no caso deste projeto, as informações repassadas eram apenas as relacionadas ao infravermelho, ou seja, o relatório é configurado para conter somente os dados de posições dos LEDs e informações básicas do controle, tais como: o controle está seguramente conectado, qual é o jogador, o nível de bateria restante, informações de posição relativas aos focos de luz nos eixos x e y, e se os pontos estão ao alcance da câmera.

Um das dificuldades de implementação foi como inferir a distância no eixo de profundidade (eixo z) uma vez que a câmera é um dispositivo bidimensional e o *wiimote* não possui recursos que forneçam esse tipo de informação. Neste trabalho desenvolveu-se um modelo baseado na proporção de distâncias.

Os cálculos foram realizados da seguinte maneira: primeiramente o sistema foi calibrado, colocando a barra sensora à uma distância de 1 metro do controle e sabendo a distância original real, em metros, entre os LEDs da barra sensora, é possível obter a informação de distância virtual dos pontos, fornecida em *pixels* pela biblioteca. Com este valor, através de uma triangulação simples, estimou-se a distância do eixo z à barra sensora, dentro do mundo virtual. Desta forma criou-se uma escala *pixels/m* para esta conversão.

A maneira como modelamos essas aproximações estão explicitadas nas formulas:

- *DiatânciaDeCalibragemDoSistema*: distância do *wiimote* até a barra sensora

$$(i) X1 \text{ câmera} = \frac{\text{distXdosPontosDeLuzNaBarraSensora}}{(\text{distXDosPontosNaTela} * \text{diatânciaDeCalibragemDoSistema})}$$

$$Y \text{ câmera} = \frac{\text{distYdosPontosDeLuzNaBarraSensora}}{(\text{distYDosPontosNaTela} * \text{diatânciaDeCalibragemDoSistema})}$$

$$(ii) X_{\text{medio}} = (X1_{\text{camera}} + X2_{\text{câmera}}) / 2$$

$$Y_{\text{medio}} = (Y1_{\text{camera}} + Y2_{\text{câmera}}) / 2$$

$$(iii) \text{anguloX} = \frac{(\text{pontoCentralX} - (1024/2)) * \text{escalaPixelsParaGraus}}{\text{escalaPixelsParaGraus}}$$

$$(iv) \text{deltaAnguloX} = \text{anguloXAnterior} - \text{anguloX}$$

Com a fórmula (i) é possível calcular a distância real do observador, fazendo uma proporção das distâncias em tela com a distância em metros.

Em (ii), encontra-se o ponto médio, em *pixels*, e com a diferença dos ângulos nos eixos x e y, obtêm-se o deslocamento feito, fazendo uso de uma escala que reflete a medida em graus reais em *pixels*. Para isto utilizamos as fórmulas (iii) e (iv).

Cabe ressaltar que toda a calibração foi feita com medidas aproximadas, sem considerar a verdadeira distância focal da câmera e possíveis distorções radiais. Contudo os resultados obtidos foram muito satisfatórios.

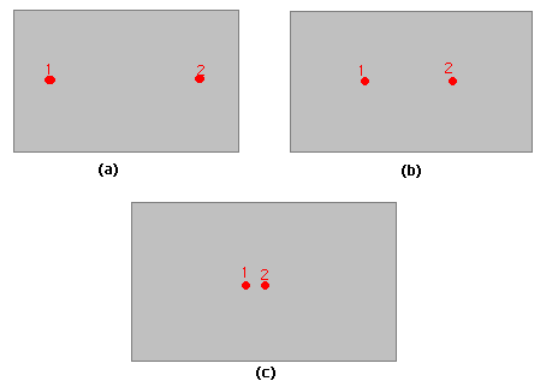


Figura 3:

- (a) observador próximo
- (b) observador à uma distância média
- (c) observador muito afastado

O modelo câmera possui limitações. É fundamental que a barra sensora se encontre em um plano paralelo à tela, e consequentemente à câmera. Utilizando somente um *wiimote* é impossível identificar a rotação da cabeça do usuário, pois quando isto ocorre, a câmera, por estar em um plano bidimensional, capta apenas a aproximação dos pontos, confundindo-se assim com o afastamento do usuário. A explicação para esse fato, fica visível na figura 3.

A aplicação desenvolvida neste trabalho consiste em uma imagem inserida como textura na face interna de um cilindro e uma câmera posicionada no centro do mesmo. Através do posicionamento do usuário, a câmera é movimentada de maneira que proporcione ao

usuário uma visão panorâmica da imagem gerada. Esta aplicação é semelhante a proposta por [6], tendo no entanto um controle mais intuitivo.

Para criar o modelo de profundidade do sistema, utilizaram-se os cálculos feitos anteriormente, de acordo com as medidas de distância entre os pontos. Quando ocorre uma aproximação da barra sensora em relação ao *wiimote* fixo, tem-se que os pontos se afastam na visão da câmera, como se pode ver na Figura 3.a. Desta forma o programa deduz que ocorreu uma aproximação do observador, o que se traduz na aproximação da câmera em relação ao panorama cilíndrico. De forma semelhante, quando o observador se afasta do controle ocorre uma aproximação dos pontos projetados, conforme se observa na Figura 3.c, gerando-se assim uma aproximação virtual dos pontos. Com isto reposiciona-se a câmera no eixo z, seguindo uma interpolação linear em relação à movimentação do observador.

Para o sistema de rotação, através da distância do ponto central (diferença entre os dois pontos) ao centro da tela, calcula-se o ângulo onde o usuário se encontra posicionado, rotacionando-se a câmera de acordo com esta posição, para modificar o panorama da imagem.

Na figura 4, pode-se visualizar a aplicação implementada. Os pontos vermelho e amarelo são captados pela câmera, enquanto que o ponto verde é o ponto central. No canto superior esquerdo pode-se ver dados aproximadamente calculados para movimentação da câmera.



Figura 4: Panorama com dados relativos à posição do observador.

6. Conclusão e Trabalhos Futuros

A inovação do sistema proposto consiste em implementar um componente de câmera e um de tratamento de entrada de dados em XNA, de forma que estes possam ser totalmente reutilizáveis e acopláveis a qualquer outro projeto nesta linguagem. Para verificar os resultados, implementou-se um protótipo em XNA, lançando-se mão de recursos gráficos interativos.

Vale ressaltar também que a forma como foram efetuados os cálculos de distância e ângulos do observador ao *wiimote* foi totalmente diferente da abordagem utilizada por [1], sendo que se utilizou um

novo método, com resultados bastante próximos aos ideais.

Como uma extensão do trabalho sugere-se adicionar uma nova funcionalidade através de um novo LED infravermelho posicionado na ponta do dedo indicador do usuário. O mesmo poderá apontar para uma determinada posição na tela, fazendo com que a câmera posicione-se exatamente neste local.

Como a câmera do *wiimote* capta até quatro pontos, não haverá restrição tecnológica para este passo.

Assim, pré-definindo “âncoras” que poderão ser apontadas, basta somente identificar a posição deste no panorama e verificar qual a nova imagem de panorama mais próximo do mesmo que está disponível, para o reposicionamento da câmera.

Um problema encontrado no projeto e que dificultou a implementação, foi o pouco alcance da câmera. Os experimentos realizados apontaram que o ângulo máximo de abertura captado no plano horizontal é de apenas 19 graus no lado direito e 22 graus no lado esquerdo. No plano vertical há uma pequena diferença de abertura, 14 graus para cima e 17 graus para baixo. Sugere-se que utilizando mais de uma câmera é possível diminuir este fator de erro.

Uma outra possibilidade é o uso de dois *wiimotes* funcionando como um par de câmeras estereó, o que possibilitaria um cálculo mais preciso da profundidade do observador.

7. Referências

- [1] LEE, J.. **Head Tracking for Desktop VR Displays using the Wii Remote**. Disponível em <http://www.cs.cmu.edu/~johnny/projects/wii/>. Acesso em 10 abr. 2008.
- [2] GABOR, D.. Holography, 1948-1971. **Science**, v. 177, n. 4046, p. 299 – 313, jul. 1972.
- [3] PROTÓTIPOS de TV Holográfica. Disponível em www.geocities.com/doctorlunazzi/protTV/protTV.htm. Acessado em 22 maio 2008.
- [4] UBBERCOOLHOME Sells Unique 80-Inch Holographic Display. Disponível em <http://www.electronista.com/articles/06/11/21/80.inch.holograph.display/>. Acesso em 20 maio 2008.
- [5] PEEK, B.. **Managed Library for Nintendo's Wiimote**. Disponível em www.codeplex.com/WiimoteLib. Acesso em 10 abr. 2008.
- [6] CHEN, S.. **Quicktime VR - An Image-based Approach to Virtual Environment Navigation**. In: ACM SIGGRAPH Computer Graphics Proceedings, p. 29-38, 1995.

Algoritmo dos Retângulos: um *pathfinding* para jogos cliente-servidor

Eduardo D. Lima Christiano L. Santos*

Universidade Federal de Sergipe, DCOMP, Brasil

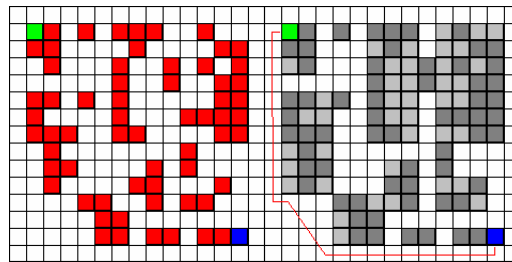


Figura 1: O Algoritmo dos Retângulos em um mapa de jogo convencional.

Abstract

This paper describes the creation of a client-server pathfinding algorithm for games as a faster alternative to traditional algorithms used. It was created looking the minimum possible complexity, targeting not the best way, but any good possible way. Therefore, it will be able to be implemented in servers, helping and monitoring the clients.

Resumo

Esse artigo descreve a criação de um algoritmo de *pathfinding* voltado para jogos cliente-servidor como uma alternativa mais rápida aos algoritmos tradicionais usados. Ele foi criado de uma forma que o seu tempo de execução seja o mínimo possível, visando não o melhor caminho, mas sim encontrar um bom caminho possível. Sendo assim, ele será capaz de ser implementado em servidores, auxiliando e policiando os diversos clientes.

Palavras chave: pathfinding, jogos, inteligência artificial, mapa, cliente-servidor

Contatos:

doriaeduardo@gmail.com

*christianolimasantos@yahoo.com.br

1. Introdução

Algoritmos de *pathfinding*, além de outras, possuem uma grande utilidade em jogos de computadores. Praticamente todo jogo possui personagens não controlados por humanos que precisam ser capazes de se deslocar pelo mapa chegando de um ponto a outro e demonstrando assim ter comportamento aparentemente inteligente. Até mesmo em personagens controlados por humanos, existem situações onde o usuário não o controla passo a passo, ele apenas escolhe um ponto no mapa e o jogo direciona-o até determinado ponto, seguindo uma trajetória correta e às vezes até a mais rápida de todas [Buckland 2002; Rabuske 1995].

Geralmente um algoritmo de *pathfinding* possui uma alta complexidade. Isso acontece devido a uma grande quantidade de informações que devem ser processadas e à enorme possibilidade de caminhos que ele pode seguir. Um fato interessante é que a grande maioria dos algoritmos procura sempre o melhor ou mais rápido caminho, dando a impressão de que o personagem do jogo possui uma capacidade de raciocínio sobre-humana.

Nesse artigo é apresentado o Algoritmo dos Retângulos, que foi desenvolvido visando ter um custo computacional muito abaixo dos algoritmos tradicionais, porém com resultados satisfatórios. É demonstrado também seu uso em jogos cliente-servidor e como ele pode ser mais eficiente que outros.

2. Estado da Arte

Muitos fatores devem ser levados em conta quando se escolhe um algoritmo de *pathfinding* no desenvolvimento de um jogo de computador. Um dos fatores principais que afetam a “inteligência” do algoritmo é o tipo do mapa. O desenvolvedor de jogo precisa saber exatamente qual o tipo de mapa que seu jogo terá. Se forem mapas no estilo labirintos, ou mapas com objetos pequenos e esparsos, como vegetação, ou até construções um pouco mais densas. Outro fator importante é o número de vezes que o *pathfinding* é usado – alguns jogos possuem uma infinidade de personagens não controlados por humanos e poucos personagens controlados, em outros jogos isso se inverte. Logo, tudo influencia no tipo do algoritmo que deve ser usado. Se ele deve ser mais complexo e menos eficiente computacionalmente falando ou se deve ser mais simples e muito mais eficiente [Rabuske 1995].

2.1 Método A*

A solução mais famosa e usada atualmente nos jogos é o algoritmo A*. Ele não possui uma grande complexidade no caso médio e é simples de ser

implementado. Seu funcionamento baseia-se em achar o melhor vizinho para chegar a determinado destino [Rich e Knight 1993].

Num jogo de computador geralmente o personagem se encontra em um mapa dividido em blocos e esses blocos são separados em dois tipos: os livres e os bloqueados. Nos blocos livres o personagem pode caminhar, nos bloqueados ele não pode, pois ali existe algum objeto, como paredes ou outros personagens. Essa divisão em regiões é feita a fim de melhorar o desempenho do algoritmo, pois se a análise e cálculo fossem feitos pixel a pixel a precisão do movimento seria ideal, porém o tempo de processamento do algoritmo seria muito maior.

Desta forma, a maior parte dos jogos convencionais a movimentação do personagem em oito direções (nos eixos vertical, horizontal e diagonais), permitindo assim que, a partir de um dado bloco do mapa possa chegar a outro.

O algoritmo A* determina a melhor opção por meio do processamento e combinação de dois elementos: o custo para chegar ao quadrado vizinho e o custo estimado (heurística) para chegar deste ao destino [Lester 2005].

2.2 Método Dijkstra

Esse método é menos usado que o A*, porém ainda assim é outro dos mais usados em jogos. Ele tem um funcionamento semelhante ao A* e ambos utilizam grafos como estrutura de dados.

O funcionamento do Dijkstra se baseia em tentar achar o caminho mais curto entre dois vértices do grafo. Para fazer isso ele monta toda a estrutura do mapa pela descrição geométrica, criando um grafo com o peso de cada aresta representando as distâncias. Em seguida, ele através do grafo, calcula a menor distância que deve ser percorrida para chegar do vértice origem ao vértice destino [Graham et al. 2003].

3. O Algoritmo dos Retângulos

Normalmente em jogos cliente-servidor, o servidor precisa ficar monitorando os passos dos clientes a todo o momento: sempre haverá pessoas com más intenções de trapacear no jogo e cabe ao servidor impedir esse tipo de ação. Algumas maneiras de saber se um jogador movimentou seu personagem de um ponto a outro sem trapacear é: informando sua posição passo a passo ou implementando o próprio algoritmo de *pathfinding* usado no cliente, também no servidor. O primeiro caso exige um grande consumo da banda, já o segundo requer um maior processamento e consumo de memória no servidor. Cabe então ao desenvolvedor escolher a melhor alternativa levando em conta os três fatores: memória utilizada, banda necessária e tempo de processamento.

O Algoritmo dos Retângulos é uma alternativa aos outros algoritmos de *pathfinding* e foi desenvolvido para ter a mínima complexidade computacional com uma boa qualidade de soluções. Com sua complexidade baixa, ele torna possível a sua implementação em servidores sem requerer tanta capacidade de processamento.

Outro fato interessante é que diferentemente da maioria dos algoritmos, ele não acha o melhor caminho sempre, mas sim procurava um bom caminho, de forma similar a como um ser humano procuraria, tornando assim o comportamento obtido mais realista.

3.1 Procurando o Destino

Como foi dito anteriormente, o mapa deve ser dividido em regiões (blocos) iguais, e essas regiões são de dois tipos: livres e bloqueadas. A ideia inicial desse algoritmo é conhecer o ponto no mapa aonde se quer chegar. Sabendo isso, o personagem sai de sua posição inicial e segue para o próximo bloco vizinho que seja mais próximo do bloco destino. Esse procedimento é feito sucessivamente até achar alguma barreira.

Quando o próximo bloco vizinho escolhido for um dos considerados bloqueados, o mesmo não poderá fazer parte do caminho. O personagem precisa então, antes de seguir, saber um modo de desviar desse bloqueio. A ideia do algoritmo para saber a melhor direção é “olhar” para os lados e ver qual é o lado mais curto para sair desse bloqueio. No caso da Figura 2, percebe-se claramente que a melhor direção a seguir é para baixo, pois o personagem precisa atravessar dois blocos para baixo, contra seis para cima.

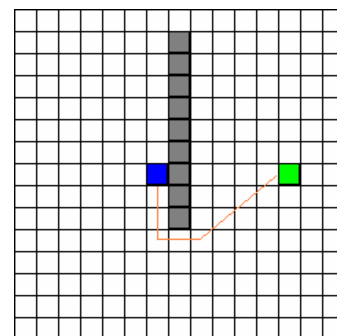


Figura 2: Personagem (azul) tentando chegar ao destino (verde)

3.2 Formando Retângulos

O procedimento descrito anteriormente funciona muito bem, porém em alguns casos ele não vai saber “sair de uma barreira”. Esses casos ocorrem quando tanto um lado como o outro dela não estão livres. Observando a Figura 3 consegue-se notar que ambos os lados da barreira estão bloqueados, logo o personagem não consegue calcular o melhor lado para sair ficando, dessa forma, parado.

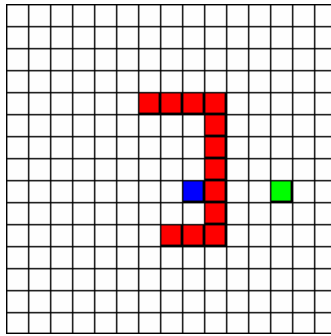


Figura 3: Personagem (azul) tentando chegar ao destino (verde)

Casos como esse devem ter um tratamento diferente: o algoritmo precisa fechar todos os objetos formando retângulos, preenchendo-os internamente e eliminando a possibilidade de o personagem entrar neles. O resultado do procedimento pode ser observado na Figura 4. Nesse caso o personagem não vai nem chegar a entrar no objeto, pois ele fará uma trajetória contornando a parte externa.

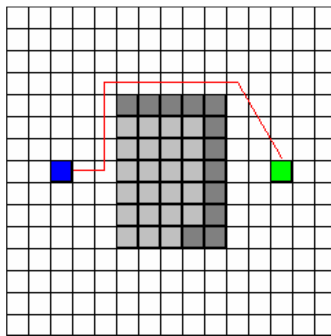


Figura 4: Personagem (azul) indo ao destino (verde)

3.3 Saindo dos Retângulos

Praticamente todos os tipos de mapa podem ser resolvidos até agora, porém existem situações em que, ou o personagem ou o destino estão em uma área que vai ser usada para formar um retângulo. Esses casos devem ser resolvidos encontrando a saída/entrada de cada retângulo. A saída/entrada deve ser o ponto fora do retângulo mais próximo do destino. Como pode ver na Figura 5, o personagem sabe exatamente para onde seguir.

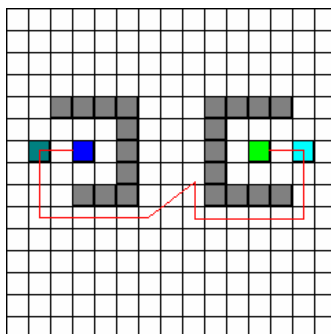


Figura 5: Personagem (azul) precisar sair do seu retângulo para chegar ao destino (verde)

Para evitar cálculos desnecessários, só são calculadas as saídas/entradas dos retângulos em que

estão o personagem ou o ponto de chegada. Além disso, quando o personagem estiver na saída do seu retângulo, ele deve ser proibido de voltar a entrar, o que o obriga a dar a volta e ir em direção à entrada do próximo retângulo, pois caso ele entre novamente, ficaria preso.

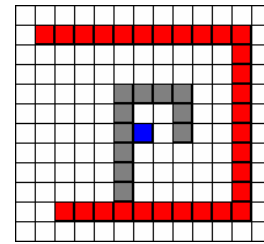


Figura 6: Exemplo de retângulo recursivo

Na Figura 6 é possível observar que o retângulo principal (em vermelho) está encobrendo outro retângulo menor (em cinza). Se o algoritmo fosse feito verificando apenas os pontos adjacentes para formar um retângulo, ele formaria um retângulo só, bem grande, nesse mapa e dependendo da posição do personagem, ele talvez nunca conseguiria chegar ao ponto de destino (em azul). O que deve ser feito é usar esse mesmo algoritmo, porém recursivamente. Assim que achar o primeiro retângulo (em vermelho) eliminá-lo e faz o mesmo procedimento para os retângulos internos, até não sobrar mais nenhum. No exemplo da imagem, dois pontos de saída/entrada serão encontrados: um referente ao retângulo vermelho e outro referente ao retângulo cinza.

Os pontos de entradas/saídas encontrados devem ser adicionados a uma pilha dependendo da ordem que vão ser buscados. Cada ponto atingido pelo personagem deve ser eliminado da pilha, forçando-o a buscar o próximo ponto. Essa busca só é terminada quando não existem mais pontos de entrada/saída na pilha, lembrando que o último ponto é o destino final do personagem.

3.4 Solucionando os Mapas

De acordo com as técnicas supracitadas, o algoritmo provavelmente seria capaz de solucionar todos os mapas de objetos esparsos. Mapas no estilo labirinto, onde muitas paredes são adjacentes umas às outras, o algoritmo encontraria muitos pontos de entrada/saída que talvez confundissem o personagem. Em outros casos, mesmo mapas grandes, porém com objetos esparsos, como os observados em muitos jogos massivamente multi-usuários, o Algoritmo dos Retângulos resolveria muito bem.

O fato de o Algoritmo dos Retângulos ser recomendado para usar em servidores é que todo o processamento da formação dos retângulos das partes fixas do mapa pode ser feito antes de iniciar a partida. Desta forma, se nem o personagem nem o local destino estiverem dentro de algum retângulo já formado, o algoritmo faria somente uma busca simples. Caso algum deles esteja dentro de algum retângulo, o

procedimento é feito conforme fora explicado, identificando as saídas/entradas recursivamente e colocando-as na pilha.

3.5 Unidades Móveis

O procedimento usado para desviar das unidades móveis do mapa (outros personagens, veículos, etc.) é praticamente o mesmo daquele usado acima nas partes fixas. A principal diferença é que o cálculo das regiões dos retângulos não poderá ser feita com antecedência.

A idéia nesse caso é deixar o personagem andar pelo mapa independente da posição das partes móveis, desviando-se de todos os objetos que tiverem a forma de retângulo, pois os cálculos já foram feitos. Quando encontrar algum objeto que não consiga passar, provavelmente esse objeto é formado por partes móveis. A partir desse momento, pega como referência o ponto de colisão e é feito o mesmo cálculo dos retângulos descrito acima. Dessa forma, os novos pontos de entrada/saída são adicionados à pilha.

Usando essa idéia, o personagem pode adquirir uma postura não muito inteligente, pois quanto às unidades móveis, ele primeiro colide para depois descobrir como desviar. Porém, levando-se em conta que no tipo de jogo em que este algoritmo será empregado, as partes móveis raramente passam de um bloco único no mapa e quando passam ocupam áreas retangulares, o resultado do algoritmo será pouco afetado.

3.5 Outras Soluções

O Algoritmo dos Retângulos foi desenvolvido para ser eficiente e, dessa forma, ser usado além dos clientes, também em servidores, policiando e monitorando as trajetórias de seus usuários, como já foi dito. Nos clientes, algumas abordagens diferentes podem ser usadas. Uma delas é o cálculo dos retângulos sendo feito em tempo real. Às vezes não é vantagem gastar um tempo maior fazendo o pré-processamento do mapa completo, então só é calculada a parte que vai ser usada em determinado momento.

Uma boa alternativa de cálculo dos retângulos em tempo real é a “tentativa e erro”. O algoritmo faz uma trajetória pelo mapa até encontrar um obstáculo que não consegue passar. Caso encontre, é feito o cálculo do retângulo que pertence ao ponto que foi colidido. Depois volta até o ponto inicial e refaz a trajetória. Esse procedimento só termina quando a trajetória chega ao ponto final, fazendo assim o caminho correto, fazendo o personagem andar sobre ele.

Um fato interessante a ser observado é o formato da trajetória, uma vez que o personagem somente desvia de um obstáculo quando o toca. Isso às vezes deixa o movimento muito artificial, como pode ser observado na Figura 7.

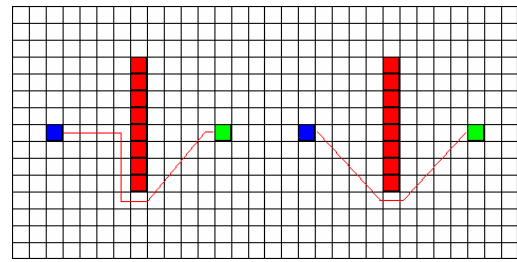


Figura 7: Na esquerda a trajetória normal, na direita a trajetória ideal

Uma solução para esse problema seria fazer a trajetória em ambos os sentidos: do início para o fim e do fim para o início. Posteriormente selecionam-se os pontos mais eficientes de cada uma. Desse jeito teria uma trajetória mais próxima daquela que seria escolhida por um humano.

4. Conclusão

Pôde-se observar que o Algoritmo dos Retângulos é uma alternativa rápida e eficiente se comparada aos algoritmos tradicionais. É fácil observar que seu caso médio vai depender diretamente do mapa usado no jogo. Porém, em muitos casos, será uma busca simples, o que permite sua implementação eficiente nos servidores. Deste modo, será possível controlar passo a passo a movimentação dos personagens, sabendo se houveram fraudes no movimento realizado por eles.

Além disso, o Algoritmo dos Retângulos também se mostrou como uma boa alternativa na movimentação de personagens não controlados por humanos, podendo ser empregado, inclusive, em outros tipos de jogos.

Como futura extensão, espera-se aplicar o algoritmo desenvolvido em um projeto de jogo massivamente multiusuário e estudar os resultados obtidos.

Referências

- BUCKLAND, MATH, AI Techniques for Game Programming. Game Developer Series, Premier Press, 2002.
- GRAHAM, ROSS, McCABE, HUGH, SHERIDAN, STEPHEN, Pathfinding in Computer Games, ITB Journal, 2003.
- LESTER, PATRIC., 2005. A* Pathfinding para Iniciantes. Traduzido por Raynaldo N. Gayoso. Disponível em: <http://www.policyalmanac.org/games/aStarTutorial_port.htm>. Acesso em: 18 AGO. 2008.
- RABUSKE, RENATO A., Inteligência Artificial, Editora da UFSC, 1995.
- RICH, ELAINE, KNIGHT, KEVIN, Inteligência Artificial, MAKRON BOOKS, 1993.

GAME ENVIRONMENT

Um novo conceito em Design de Interiores



Fig. 1 - Jogo Resident Evil 4 (CAPCOM, 2005), versão GameCube (Nintendo)

Resumo

Este artigo descreve como os profissionais habilitados em design de interiores agregam um valor simbólico no desenvolvimento de um jogo eletrônico, tal como no processo de construção de um ambiente virtual, de conceitos e ferramentas até então somente empregados na projeção e organização de um ambiente real.

Palavras Chaves: design, design de interiores, ambientes virtuais, narrativas.

Contato:

janielversiane@yahoo.com.br
virgiliodmaia@yahoo.com.br

1. Introdução

O design, enquanto área de conhecimento e de produção, destaca-se pela sua multiplicidade de manifestações, de tal modo que, em toda e qualquer forma de representação, pode-se ver design.

Assim, o design “é uma atividade criativa que consiste em determinar as múltiplas facetas dos produtos, processos, serviços. O design é um fator chave de humanização das tecnologias e troca econômico-cultural” [Mozota *apud* Silva, 2005]. Como se percebe, o design nasce de uma idéia, que discutida, planejada e orientada por um adequado processo de produção, possibilita o alcance do resultado final, que pode ser um produto, uma embalagem, uma peça gráfica e, porque não, todo um projeto de organização de um ambiente interno ou externo, como um escritório ou um jardim. O design é, portanto, versátil, apresentando-se plural e multidisciplinar.

Desse modo, entre as diferentes formas de manifestação do design, encontra-se o design de interiores. Este, posicionando-se entre a arquitetura e as artes plásticas, possibilita uma atuação profissional orientada para a necessidade de se projetar os espaços, interferindo positivamente no arranjo dos ambientes internos e externos. Nesse sentido, [Vinícius Alberto Moraes, 2007] afirma que “a importância da prática do design de interiores é causar efeito no ambiente construído e em seus usuários”. O designer de interiores é, pois, o profissional que, relacionando as características do ambiente com as necessidades do usuário, consegue organizar as formas dentro de um recinto, construindo, para cada cliente, uma concepção única e nova

de espaço. “O design em si é a chave para a elaboração do projeto. É ele que imprime estética e funcionalidade ao espaço. A partir das necessidades do ambiente construído, ele se apropria de valores intangíveis, determinantes para a captação dos desejos dos usuários” [Moraes, 2007]

Entretanto, em tempos de globalização, essa visão clássica do design de interiores parece dar lugar a uma nova perspectiva de atuação profissional, sincronizada com as demandas de um mercado em acelerada expansão: o mercado virtual. Assim, a imagem de um profissional exclusivamente dedicado ao desenvolvimento de identidade visual arquitetônica, de composição e decoração de ambientes reais, internos e externos (residências, escritórios e jardins), gradativamente, divide espaço com um novo tipo de profissional que, aplicando os conceitos e ferramentas do design de interiores, consegue criar, em um ambiente virtual, as feições e características de um ambiente real. Nesse sentido, como ocorre em outras profissões, o design de interiores se expande no ritmo acelerado das transformações da vida globalizada, em que novas idéias e novas tecnologias surgem, a cada instante, para atender as demandas crescentes do homem moderno. São por essas e outras circunstâncias que os profissionais formados em design de interiores desenvolvem projetos inovadores capazes de surpreender até mesmo os usuários de um ambiente virtual, com destaque aos usuários de jogos eletrônicos.

Nesse sentido, nota-se que, por detrás do desenvolvimento de um game, há sempre uma equipe criativa e inovadora, na qual o profissional em design de interiores, atuando em cada uma das etapas de desenvolvimento, faz a diferença, sobretudo no que se refere à construção dos *backgrounds* (cenários) dos respectivos ambientes delimitados no *Game Design Document (GDD)* – documento, no qual se encontra a idéia inicial do jogo, com a determinação da narrativa, características dos personagens e dos cenários, bem como informações relacionadas à interatividade com o jogador e limitações das plataformas em que o jogo será gerado [Clua, 2007]. Como se observa, estes, acrescidos de outras características, constituem os critérios que, no processo de construção de cenários de um game, especificam a atuação do designer de interiores.

Assim, se na projeção voltada para um espaço concreto, o designer de interiores deve-se ater às características do usuários, a fim de que melhor interajam com o espaço construído, igualmente, na projeção de um cenário virtual para games, deve o designer de interiores considerar o perfil de seus usuários, a fim de promover uma maior interação entre estes e a história narrada no game. Isso porque a forma

como os jogadores percebem o espaço virtualmente construído constitui fator determinante para a boa aceitação do jogo diante de seu público-alvo. “O designer utiliza as coisas e os objetos dentro do espaço como códigos e os usuários, com suas necessidades e sentidos, irão decodificar a mensagem contida no espaço” [Morais, 2007]. Esse raciocínio pode ser comprovado a partir de análise dos cenários do jogo *Resident Evil 4* (Fig. 1), os quais se agrupam em dois espaços diferentes: uma catedral e um castelo, ambientes de estilo marcadamente gótico. A escolha desse estilo pela equipe de desenvolvimento do jogo corresponde, como se pode ver a seguir, aos traços narrativos de *Resident Evil 4*.

2. Narrativas do jogo Resident Evil 4

A narrativa do jogo em estudo consiste no seguinte:

Em 1998, as atividades secretas da Umbrella Corporation dentro de Raccoon City chegaram ao fim. Após uma investigação, conduzida pelo governo dos Estados Unidos da América, vários diretores da Umbrella são indiciados e processados. O governo suspende indefinitivamente os negócios da Umbrella, causando sua falência.

Passaram seis anos, após os acontecimentos em Raccoon City.

Leon Kenedy (agente exclusivo do governo dos USA) é mandado em uma missão para resgatar Ashley Graham, a filha do presidente, que foi seqüestrada por um culto misterioso, e descobrir o que aconteceu com o outro agente, enviado anteriormente para resgatá-la. Leon viaja a um pequeno vilarejo rural na Espanha, chamado Pueblo, onde ele encontra uma horda de aldeões hostis que dão suas vidas para Los Iluminados, o culto que seqüestrou Ashley.

Durante as buscas, Leon descobre que eles são responsáveis pela morte do agente que desapareceu. Todos são membros de uma seita de fanáticos religiosos, tendo como chefe da seita é Lord Sadler, que é também responsável pelo rapto de Ashley. O objetivo de Sadler era sequestrar a filha do presidente com a finalidade de obter um resgate, já que Ashley havia sido deliberadamente infectada com um ovo e, assim, uma vez de volta à sua casa, iria espalhar a infecção [Wikipedia, 2008].

3. Design de interiores e Games: Ambientes virtuais e narrativas

Como se pode observar, o estilo gótico aplicado nos cenários de *Resident Evil 4* corresponde as características da narrativa do jogo (Fig. 2), pois favorece a percepção de um espaço fúnebre, sombrio, cheio de armadilhas [Gympel, 2001], capaz de prender a atenção do jogador. A esse respeito, [Paulo Oliveira, 2007], discutindo a atuação dos designers de interiores na produção de um jogo eletrônico, faz a seguinte observação: “Pesquisas históricas para adequar os ambientes aos estilos da época em que o game se passa, mobiliários específicos, materiais utilizados na época, arte específica, iluminação condizente com a realidade ... Uaw! E quando são games de guerra onde temos uma arquitetura, um urbanismo destruído pelas bombas e que faz-se necessário uma cenografia nessa linha ... outros já primam pela riqueza e luxo, um mundo perfeito onde não existem pobres ... quem sabe aqueles futuristas, visando a projeção dos sonhos humanos sobre a era espacial vindoura, a guerra dos mundos ... Talvez aqueles na linha Doom, de mocinho e bandido que sempre tem a a mesma cara, os mesmos galpões, as mesmas

fortalezas, as mesmos caixotes espalhados nos mesmos lugares ... E isto tudo adequado ao roteiro do game”.

Assim é, na relação entre o espaço, o designer de interiores e o jogador, que centra-se o estudo que se pretende realizar através deste projeto.

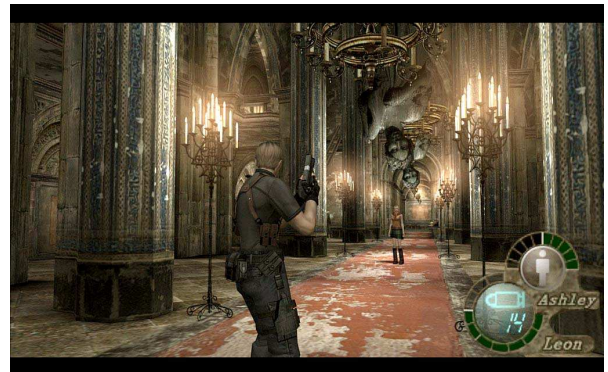


Fig. 2: Resident Evil 4 (Capcom, 2005)

4. Material e Métodos

Para o alcance do objetivo acima elencado será empregado, como método de abordagem, o dedutivo, pois, a partir do estudo dos conceitos e ferramentas aplicados pelo designer de interior na projeção de um ambiente real, será possível examinar a atuação desse profissional no processo de construção de cenários para jogos eletrônicos (ambiente virtual). Como técnica de pesquisa, será adotada a bibliográfica, através do estudo de livros, artigos, periódicos e material encontrado na internet.

5. Discussão

O design, por sua multiplicidade de manifestações, deve ser visto sob diferentes olhares, entre estes, destaca-se o que se refere à atuação do designer de interiores voltada para a projeção de ambientes virtuais que, no mundo globalizado, afirma-se como uma nova segmentação da linguagem projetal. Essa nova segmentação chama atenção quando aplicada no processo de desenvolvimento de um jogo eletrônico, no qual contribui o designer de interiores por meio da diferenciação do produto final. Tal diferenciação é alcançada a partir da aplicação, no processo de construção de um ambiente virtual, de conceitos e ferramentas até então somente empregados na projeção e organização de um ambiente real.

Dessa forma, é o designer de interiores o profissional habilitado, técnica, humanística e artisticamente, para agregar valor simbólico ao jogo produzido, favorecendo, em relação ao produto final, uma percepção positiva do usuário. Assim, o trabalho a ser desenvolvido justifica-se pela necessidade de se demonstrar uma nova possibilidade de atuação do designer de interiores, ampliando, com isso, os horizontes de uma profissão que por si mesma é rica de manifestações, múltipla e plural.

6. Resultados

Na busca de uma melhor interpretação sobre a participação desse profissional, observaremos em quais as áreas os

designers de interiores poderão ser de grande auxílio no desenvolvimento de um jogo eletrônico. Seja ela na equipe de game design, level design, arte/animação e programação.

Para o alcance desses resultados, analisaremos em detalhes todas as etapas de produção de um game, discriminando as principais características de cada área em estudo, dando ênfase ao designer de interiores, que por contribuição ajudará na criação dos ambientes virtuais dos jogos produzidos.

6.1 Game Design

Compreende a construção do projeto original do jogo, com suas respectivas diretrizes de execução, incluindo o perfil dos personagens, jogabilidade, narrativas, detalhes das missões e níveis e as condições dos cenários (cor, textura, sombra, luminosidade, profundidade, perspectiva, movimentação, interatividade); e durante esta fase, é elaborado o *Game Design Document* [Perucia et All, 2005], um documento que descreve as características citadas anteriormente em detalhes. O *Game Design Document* funciona como um roteiro de cinema. Com base nessas informações, os artistas irão criar o visual e os programadores desenvolverão a interatividade do produto.

6.2 Level Design

Nesta etapa, o processo de desenvolvimento é imprescindível em projetos de grandes complexidades, com muitas fases e missões. Nesse sentido, o *Level Design* é um mapa geral, com desafios e missões que o jogador deve cumprir para vencer cada fase [Perucia et All, 2005]. O seu principal objetivo é desafiar o raciocínio lógico do jogador, utilizando-se de métodos que permitem uma ligação direta entre o jogo e jogador. Após a finalização do level design, os artistas e programadores, poderão se dedicar à produção das fases.

6.3 Arte e Animação

Esta equipe é responsável pela parte visual do game, pelas interfaces gráficas e animação. Neste momento há dois profissionais envolvidos, os artistas de conceito e os animadores.

Os primeiros desenvolvem cenários, personagens e itens, através de desenhos (rascunhos), conforme uma análise minuciosa do roteiro elaborado pelo game designer. Após a realização deste trabalho, o resultado do mesmo será submetido aos animadores que digitalizarão essas imagens, sendo elas 2D ou 3D, ganhando movimentos segundo uma sequência de ações (frames).

6.4 Programação

Neste momento, a equipe de programação, será responsável pela execução do jogo, utilizando linguagem de programação e inteligência artificial. O seu trabalho é unificar o projeto num produto, colocando código com arte, código com música ou código com jogabilidade. Mas no final tudo é código [LDGAMES, 2006].

7. Conclusão

As considerações do presente trabalho demonstram que, ainda há uma série de estudos e análise a serem pesquisados,

que possibilitarão melhor entendimento com relação ao profissional de design de interiores no processo de desenvolvimento de um jogo eletrônico, sendo que, a sua importância é primordial para a construção dos ambientes virtuais dos games.

Referências

- CLUA, E. W. G. Desenvolvimento de jogos 3D. In: SILVA, E. M.; MOITA, F. M. S.; SOUSA, R. P. (Orgs.). **Jogos eletrônicos: construindo novas trilhas**. Campina Grande: EDUEP, 2007.
- GYMPEL, J. **História da Architectura – da antiguidade aos nossos dias**. Colônia: Konemann, 2001.
- LDGAMES PRODUTORA DE SOFTWARES LTDA. **Metodologia de produção e desenvolvimento de um software de entretenimento. Caso: Oniria Entertainment**. Relatório de Pesquisa. Disponível em: <http://www.abragames.org/downloads.html>. Acessado em: dez. 2006.
- MORAIS, V. A. **A importância do ecodesign para o design de interiores**. Disponível em: www.anpedesign.org.br/.../pdf/A%20importancia%20do%20ecodesign%20para%20o%20design%20de%20interiores.pdf. Acessado em: jun. 2008.
- PERUCIA, A. S., BERTHÊM, A.C., BERTSCHINGER, G.L., MENEZES, R.R.C. **Desenvolvimento de jogos eletrônicos – Teoria e Prática**. São Paulo: Novatec, 2005. p. 28.
- SILVA, Claudete Barbosa da. **Design e estratégias competitivas**. T&C Amazônia, Ano III, Número 17, 2005.
- WIKIPEDIA, 2008. Disponível em: http://pt.wikipedia.org/wiki/Resident_Evil_4

Uma abordagem usando algoritmos genéticos em Agentes Inteligentes para Jogos

Felipe José Rocha Vieira Christiano Lima Santos

Universidade Federal de Sergipe, Departamento de Computação, Brasil

Resumo

É consenso que a tomada de decisões bem como o comportamento assumido pelos personagens em um jogo é uma das áreas mais discutidas em Inteligência Artificial aplicada a jogos. Este artigo tem como objetivo propor o uso de algoritmos genéticos na criação de agentes inteligentes em jogos, sendo este o passo inicial para aplicações mais complexas, bem como apresentar o FightManager, aplicação desenvolvida durante as pesquisas como estudo de caso.

Palavras Chave: Inteligência Artificial, Agentes, Algoritmos Genéticos, Máquina de Estados Finitos.

Contato dos Autores:
{felipejrveira,christianolimasantos}@yahoo.com.br

1. Introdução

Com o passar do tempo observou-se a evolução da Inteligência Artificial como forma de buscar soluções a desafios ainda não resolvidos, muitos deles presentes na área de jogos [Rabuske 1995].

No início de uma aventura em um jogo o foco normalmente está em apresentar o mundo ao jogador: história, aliados e inimigos, dentre outros, e com o desenrolar da mesma, jogador e personagem começam a se entender melhor e juntos evoluem no jogo. Desta forma, seria muito decepcionante perceber-se que apenas estes evoluíram, permanecendo o mundo ao seu redor estático, com adversários que já não representam mais dificuldades e personagens autônomos demonstrando o mesmo comportamento repetitivo [Buckland 2002].

Nesse estágio, o jogo já não apresenta mais nenhum desafio. No entanto, o inverso também é desmotivante: inimigos muito fortes podem acabar levando o jogador a abandonar a aventura. Com isto, percebe-se que é de extrema importância uma correta evolução dos NPCs - Non-Player Character (personagem autônomo) que respeite o balanceamento do jogo, sendo este o foco do artigo.

No desenvolvimento de agentes inteligentes em um cenário, deve-se ter bastante cuidado, pois os jogadores esperam que os NPCs se comportem como outros jogadores, raciocinando e agindo de forma

similar a como um jogador humano comportar-se-ia diante de uma mesma situação.

Este trabalho apresenta uma abordagem ao uso de algoritmos genéticos em agentes inteligentes a fim de desenvolvê-los de forma evolutiva.

2. Técnicas empregadas

Existem diversas técnicas de algoritmos de IA aplicadas em jogos: minimax, técnicas de pathfinding, redes neurais, flocking, entre outros.

Neste artigo propõe-se uma arquitetura para o comportamento e tomada de decisões dos personagens focada em três das diversas técnicas existentes: máquinas de estados finitos, agentes inteligentes e algoritmos genéticos.

2.1 Máquina de Estados Finitos

Uma máquina de estados finitos (MEF) é uma modelagem do comportamento de um sistema que possui um número definido e limitado de condições, alternadas de acordo com os estímulos recebidos.

Segundo Brownlee [2002], as MEF são compostas de quatro elementos principais:

- Estados – definem comportamento e podem produzir ações;
- Transições de Estado – são movimentos de um estado para outro;
- Regras ou Condições – devem ser conhecidas para permitir uma transição de estado;
- Eventos de Entrada – são gerados externamente ou internamente, e podem ativar regras, podendo conduzir a transições.

A utilização desta técnica, apesar de possuir uma natureza previsível, é muito boa, por atribuir alguma inteligência para o personagem *a priori* e devido à sua simplicidade é rápida de projetar, implementar e em sua execução.

2.2 Agentes Inteligentes

Um agente inteligente é qualquer entidade que capte dados do ambiente e atue sobre este. Para Weiss [1999] um agente inteligente deve possuir as

seguintes características:

- **Reatividade:** percepção de seu ambiente, respondendo em tempo hábil às mudanças a fim de atingir seus objetivos;
- **Pró-Atividade:** capacidade de exibir comportamento meta-dirigido ao tomar a iniciativa;
- **Sociabilidade:** interação com outros agentes (e possivelmente humanos).

Através destas características define-se que um bom agente é aquele que a partir de um conjunto de percepções toma uma boa decisão que o leve ao objetivo que deseja.

2.3 Algoritmos Genéticos

A idéia do desenvolvimento dos algoritmos genéticos partiu dos conhecimentos da biologia, através da teoria da evolução de Darwin, daí a denominação desta abordagem de evolutiva. [Sobrinho e Girardi 2003]

Nesta técnica, diversos indivíduos diferentes são gerados aleatoriamente e somente os mais adaptados sobrevivem. O fundamento básico desta técnica é criar indivíduos, pontuá-los de acordo com suas ações e, após isso, escolher os mais aptos e fazer um “cruzamento”, gerando assim novos indivíduos possivelmente mais adaptados.

Assim, a cada geração espera-se ter seres mais aptos, podendo-se dizer que aquela população evoluiu. Para melhor compreensão é importante definir alguns termos [Buckland 2002]:

- **Crossover** – taxa de ocorrência de cruzamentos entre indivíduos selecionados, possivelmente os mais aptos. Altos níveis de **crossover** podem levar mais rapidamente à resposta, no entanto corre-se o risco de perder bons resultados encontrados anteriormente;
- **Mutação** – taxa de ocorrência de uma mudança aleatória na estrutura do indivíduo. Normalmente a taxa de mutação é bastante baixa a fim de evitar a perda das características herdadas que levaram à sua escolha sem desprezar a necessidade de variabilidade genética;
- **Fitness** – pontuação que o personagem adquiriu durante a geração, utilizada para identificar quem foi mais apto.

Testes demonstram que é possível também empregar algoritmos genéticos a fim de verificar se as regras que definem o universo de um jogo bem como sua população estão balanceadas, pois o uso de tais algoritmos permite que se perceba quais as características que mais podem afetar determinadas

situações, possivelmente quebrando o equilíbrio do jogo.

3. Arquitetura do Agente

No trabalho produzido por Monteiro [2005] foram utilizados sete módulos:

- **Perception** - Módulo de eventos responsável pela filtragem dos sensores do agente;
- **Location** - Centraliza informações sobre o modelo do ambiente para o agente;
- **Long-Term Decision** - Responsável pelo planejamento;
- **Short-Term Decision** - Responsável pela execução do plano atual;
- **Motion** - Manipula aspectos de roteamento, colisão e desvios de obstáculos;
- **Animation** - Determina a animação a ser exibida;
- **Behavior** - Atua sobre o ambiente de forma reativa.

Neste trabalho, será apresentado um modelo simplificado para a arquitetura do agente com apenas três módulos:

- **Percepção** – responsável pela análise da área onde o personagem está;
- **Decisão** – através da percepção que teve do ambiente o personagem escolhe qual melhor comportamento a se assumir;
- **Ação** – Depois de decidido qual comportamento utilizar o personagem executa sua ação.

A utilização de um modelo simplificado tem como intuito aumentar a abstração em relação à visualização que se tem do agente, assemelhando-se a como se toma uma decisão, observando o ambiente, decidindo-se e executando alguma ação.

3.1 Percepção

A percepção funciona recebendo estímulos do ambiente e levando o personagem a algumas conclusões, traçando planos de ação para serem comparados na próxima etapa de processamento.

Cada ser possui um nível de percepção, que influencia suas escolhas, por exemplo, ao observar um ônibus a certa distância, aquele que tiver melhor visão terá uma resposta mais precisa e mais rápida quando perguntado sobre que ônibus está vindo.

Isto leva a concluir que cada personagem deve ter uma percepção própria que dê a ele a possibilidade de ter opiniões diferentes dos outros, dando maior diversidade ao jogo.

3.2 Decisão

Depois de colher informações do meio e fazer uma análise prévia do que está acontecendo no ambiente o personagem pode tomar uma decisão, que acontece através do cruzamento dos dados dos oponentes com as informações do agente. Por exemplo, por mais que tenha identificado que exista um oponente fraco, o fato de ter poucos pontos de vida o impossibilita de decidir perseguir o inimigo, fazendo-o preferir fugir.

O momento da decisão é aquele em que o personagem decide qual das estratégias deve seguir entre a gama de possibilidades que sua percepção gerou. Então, por melhor que seja a percepção dele o que levará a tomar uma decisão é o conhecimento que possui de si mesmo, seu estado atual e seu objetivo.

3.3 Ação

Os dois módulos anteriores mostram ações mentais, já neste, o foco está nas ações físicas, executando o que foi decidido. A partir deste ponto é que o ambiente sofre mudanças, o personagem se move, muda sua animação e demonstra seu comportamento.

A cada turno todas as três etapas são executadas, pois só através delas é que se podem executar boas ações, já que o ambiente está em constante mudança e o que era bom anteriormente pode não ser mais.

4. Seleção e Evolução dos Agentes

Como já foi apresentado anteriormente como funciona cada agente, esta seção adentra na seleção e evolução dos mesmos. Cada unidade possui características que acabam diferenciando uma das outras, mas nem sempre estas são interessantes para o ambiente onde o agente está. E é nesta situação que se aplicam os algoritmos genéticos.

Para cada sucesso, o personagem recebe uma pontuação (*fitness*) que será utilizada para a seleção dos mais aptos, possibilitando a evolução da espécie, tendo assim oponentes e aliados mais adaptados. [Buckland 2002]

Quanto mais gerações forem geradas, encontrar-se-ão personagens mais aptos, e é importante ressaltar que este número de gerações deve possuir um limite que é quando o resultado converge para um grupo de personagens considerados os mais adaptados.

Desta forma, a seleção e evolução dos agentes podem ser empregadas como forma de criar diversos oponentes apresentando cada qual seu próprio nível de capacidade de raciocínio.

Isto levará a soluções que possuem certa variação, gerando dificuldades para o jogador que, quando bem

balanceadas, representarão obstáculos com nível de complexidade bem mensurado.

5. Estudo de Caso – FightManager

FightManager trata-se de um jogo onde o usuário deve participar de lutas em arenas (áreas fechadas onde vários lutadores confrontam-se) e treinar seu personagem de forma a buscar ser o melhor.

O jogo possui uma fase não jogável de evolução que mostra combates entre NPCs, buscando, através das interações, pontuar os agentes a fim de classificá-los e identificar os mais adaptados, possibilitando a geração de novos combatentes através das informações adquiridas com as gerações anteriores.

A figura 1 apresenta a interface gráfica de FightManager, atualmente em modo texto:



Figura 1: Interface gráfica.

Para a realização dos combates é necessário que os BOTs tenham atributos que definirão as características mentais e físicas de cada um. No FightManager, um personagem é constituído das seguintes características:

- Força (F) – Atributo relacionado a quanto de dano um personagem pode causar;
- Destreza (D) – Identifica a capacidade de deslocamento e a habilidade;
- Vigor (V) – Capacidade de absorver os danos sofridos;
- Percepção (P) – Influi diretamente na análise que o personagem faz de cada oponente;
- Intimidação (I) – Nível de medo que um oponente causa a um personagem;
- Coragem (C) – Indica os pontos de vida para um NPC começar a fugir, calculado através da seguinte fórmula: $(30 - \text{coragem})/3$.

A implementação do NPC está distribuída da seguinte forma:

- Percepção – Escolhe o oponente aparentemente mais fácil de derrotar;
- Decisão – Através da primeira análise muda

seu estado para o mais adequado entre: atacar, perseguir, fugir, descansar e morto, a Figura 2 mostra como são tomadas estas decisões.

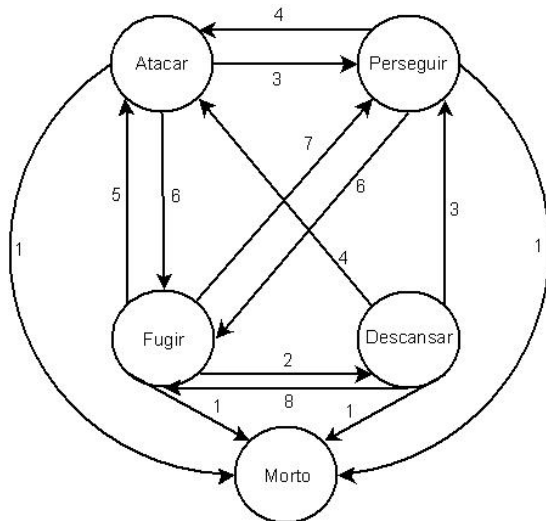


Figura 2: Todas as regras de mudança de estado.

Segue abaixo a explicação dos códigos para ocorrer transição entre os estados.

1. Pontos de Vida menor que zero
2. Oponentes estão distantes
3. Oponente alvo fora de alcance
4. Oponente alvo próximo
5. Oponente alvo em piores condições que o personagem está próximo
6. Pontos de Vida baixo e oponente alvo em melhores condições que o personagem
7. Oponente alvo em piores condições que o personagem está distante
8. Oponentes próximos;

- Ação – Executa o estado escolhido.

Ao ser executado, criaram-se inicialmente 10 personagens com os seguintes atributos:

Nome: 0 F: 7 D: 6 V: 3 P: 9 I: 5 C: 11
 Nome: 1 F: 3 D: 7 V: 8 P: 2 I: 10 C: 2
 Nome: 2 F: 7 D: 5 V: 4 P: 5 I: 9 C: 10
 Nome: 3 F: 10 D: 1 V: 6 P: 7 I: 6 C: 11
 Nome: 4 F: 8 D: 5 V: 5 P: 9 I: 3 C: 16
 Nome: 5 F: 4 D: 2 V: 10 P: 6 I: 8 C: 0
 Nome: 6 F: 10 D: 10 V: 1 P: 3 I: 6 C: 2
 Nome: 7 F: 9 D: 2 V: 2 P: 7 I: 10 C: 26
 Nome: 8 F: 4 D: 6 V: 6 P: 8 I: 6 C: 17
 Nome: 9 F: 8 D: 3 V: 7 P: 6 I: 6 C: 27

Após 30 gerações, os últimos 10 personagens tinham adotado a seguinte configuração:

Nome: 290 F: 6 D: 5 V: 8 P: 6 I: 5 C: 3
 Nome: 291 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 292 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 293 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 294 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3

Nome: 295 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 296 F: 6 D: 5 V: 8 P: 6 I: 5 C: 3
 Nome: 297 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 298 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3
 Nome: 299 F: 6 D: 6 V: 8 P: 5 I: 5 C: 3

Em outras palavras, os personagens evoluíram até a constituição que eles encontraram como sendo ideal para os combates, demonstrando assim o aprendizado propiciado bem como a eficácia do método.

6. Considerações Finais

Este trabalho é o primeiro passo para a busca de novas soluções no desenvolvimento de jogos com um melhor balanceamento, bem como personagens capazes de aprender e ter comportamentos distintos, uma vez que muitas das técnicas de IA atualmente empregadas são pouco flexíveis ou possuem taxas de aprendizado muito baixas ou mesmo inexistentes.

Outro ponto importante foi desmistificar diversos aspectos da área de jogos, permitindo assim aos pesquisadores o acúmulo de experiência na mesma.

Como futura extensão, pretende-se estudar a API gráfica OpenGL e implementar uma versão demonstração jogável do mesmo e distribuir livremente o mesmo a fim de avaliar com jogadores o potencial dos agentes empregados.

Referências

- BROWNLEE, Jason, 2002. *Finite State Machine (FSM)*. [online]. Disponível em <http://ai-depot.com/FiniteStateMachines/FSM.html>. [Acessado em 23/08/2008].
- BUCKLAND, Mat, 2002. *AI Techniques For Game Programming*. Ohio: Premier Press.
- MONTEIRO, Ivan Medeiros, 2005. *Uma Arquitetura Modular para Desenvolvimento de Agentes Cognitivos em Jogos de Primeira e Terceira Pessoa*. Universidade Federal da Bahia, Instituto de Matemática.
- RABUSKE, Renato, 1995. *Inteligência Artificial*, Editora da UFSC.
- SOBRINHO, Antonio Carlos C. da S. E GIRARDI, Maria Del Rosário, 2003. *Uma Análise das Aplicações dos Algoritmos Genéticos em Sistemas de Acesso à Informação Personalizada*. Universidade Federal do Maranhão.
- WEISS, Gerhard, 1999. *Multiagent Systems, A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. Massachusetts Institute of Technology. Editado por Gerhard Weiss, Inglaterra.
- WIKIPEDIA, *Máquina de Estados Finitos*, disponível em http://pt.wikipedia.org/wiki/M%C3%A1quina_de_estado_finito [Acessado em 25/08/2008].

Pandorga: Uma plataforma *open source* para a criação e desenvolvimento de jogos

Fábio M. Miranda, Paulo R. Lafeta, Leonardo Queiroz, Carlúcio S. Cordeiro, Luiz Chaimowicz

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

Abstract

With the growing complexity of game development, it becomes more necessary to create tools that help this task. This paper presents a new platform for the creation of games, built on top of the *Panda3D* graphic engine and the *ODE* physics engine. The platform has a programming framework, which is proposed to be easily extendable and modifiable, and also a level editor and an event editor.

Resumo

Com a crescente complexidade no desenvolvimento de jogos, cada vez mais se faz necessária a criação de ferramentas que auxiliem esta tarefa. Este artigo apresenta então uma nova plataforma para a criação de jogos, construída com base no motor gráfico *Panda3D* e no motor de física *ODE*. A plataforma possui um arcabouço de programação, que se propõe ser de fácil extensão e modificação, e também um editor de fases e um editor de eventos.

Keywords:: Panda3D, Framework, Pandorga, open source, Estrada Real Digital, ERD

Author's Contact:

{fabiom, paulo.lafeta, leoqueiroz, carlucio}@gmail.com
{chaimo}@dcc.ufmg.br

1 Introdução

O desenvolvimento de jogos é uma atividade complexa e que envolve diversas áreas, como computação, arte e roteiro. Com o intuito de facilitar o desenvolvimento, diminuir os custos e prazos, diversas ferramentas foram criadas ao longo dos anos, como motores (ou *engines*) gráficos e motores de física.

O objetivo principal deste artigo é apresentar a Pandorga: uma plataforma código aberto (*open source*) para desenvolvimento de jogos que está sendo construída em conjunto com uma nova versão do jogo Estrada Real Digital (ERD). Uma primeira versão deste jogo educacional foi apresentada em [Cordeiro et al. 2006] e [Oliveira et al. 2005].

A plataforma proposta aqui é uma camada situada entre o motor gráfico utilizado e o jogo, envolvendo também um motor de física. Ela foi construída de forma que seus componentes fossem facilmente modificados ou estendidos, podendo ser utilizada em diversos jogos. Além disso, ela também apresenta algumas ferramentas que facilitam a integração entre as diversas áreas envolvidas na criação de um jogo, como um editor de fases e um editor de eventos.

Alguns sistemas mais sofisticados e que provêm ao usuário um maior grau de abstração também têm sido utilizados na elaboração de jogos. Exemplos disso são o *Gamestudio*, *The Games Factory*, *Ray Game Designer 2* e o *Game Maker*. Estas ferramentas já possuem um motor gráfico próprio e também diversos editores, como editores de fase, script e até de modelos, como no caso do *Gamestudio*. Porém, tais alternativas não possuem o código aberto e algumas delas possuem restrições quanto à distribuição dos jogos.

A organização deste artigo é feita da seguinte maneira: na seção 2 é apresentada a plataforma e nas subseções 2.1 e 2.2 são apresentadas as principais características do arcabouço de programação e dos editores integrados. Na subseção 2.3 é feita uma discussão de como se dá o fluxo de desenvolvimento de um jogo com a adoção da Pandorga. E finalmente na seção 3 está a conclusão e as perspectivas para trabalhos futuros.

2 A plataforma proposta

A Pandorga oferece uma plataforma para a criação de jogos de maneira que vários de seus componentes possam ser reutilizados. A plataforma foi desenvolvida levando-se em consideração a orientação por objetos e *design patterns* [Gamma et al. 1995].

Toda a implementação dos recursos da plataforma foi feita utilizando o motor gráfico *Panda3D* [Goslin and Mine 2004], de autoria da *Disney* e que possui o seu código aberto. Apesar de possuir seu núcleo escrito em C++, o *Panda3D* permite a programação em Python (linguagem utilizada na Pandorga).

No início do desenvolvimento da Pandorga, o *Panda3D*, em sua versão 1.4.2, dispunha apenas de um sistema de física básico. Decidimos então adotar um motor de física separado; optamos pelo *ODE* (mais especificamente o seu *wrapper* para Python, *PyODE*) por oferecer recursos para a simulação física e testes de colisão. O *ODE* foi recentemente integrado ao motor gráfico (na versão 1.5.3), mas sua escolha inicial para a Pandorga evitará complicações com futuras versões do *Panda3D*. A utilização dos motores possibilitou que nos concentrássemos na implementação de novas funcionalidades.

O principal objetivo da plataforma é o de facilitar o desenvolvimento de um jogo. Com isso, optamos por encapsular várias funcionalidades providas pelos motores gráficos e de física. Uma visão geral pode ser vista na Figura 1.

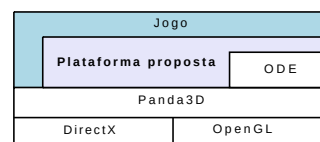


Figura 1: Plataforma proposta.

A Pandorga pode ser dividida em duas partes: o arcabouço de programação e os editores integrados. O primeiro define como será o uso das classes da plataforma, enquanto o segundo são ferramentas integradas à plataforma para auxiliar a criação dos jogos.

2.1 Arcabouço

A execução da plataforma é controlada por uma máquina de estados finitos (*finite state machine - FSM*), implementada como um *Singleton*, e que determinará em qual estado o jogo se encontra. Ao contrário dos softwares de construção de jogos citados inicialmente neste artigo, a plataforma proposta não possui dois executáveis distintos ("Jogo" e "Editor", por exemplo). O acesso aos editores é feito diretamente de dentro do jogo (*in-game*), alterando o estado da FSM para o estado de Edição de Fases ou então Edição de Eventos. A Figura 2 exemplifica as transições entre os estados.

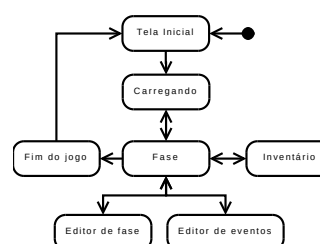


Figura 2: Máquina de estados geral do jogo.

Os estados que representam as fases do jogo, onde o personagem é controlado, herdam da classe Fase. A Figura 3 mostra algumas classes de estados já definidas na plataforma.

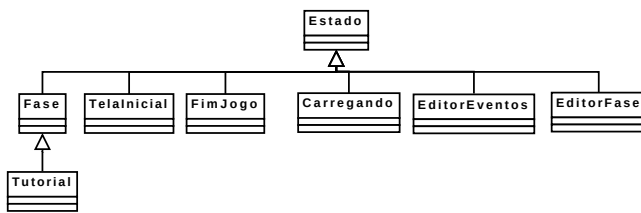


Figura 3: Classes que representam os estados do jogo.

A plataforma já implementa várias funcionalidades básicas de um jogo, como pode ser visto na Figura 4.

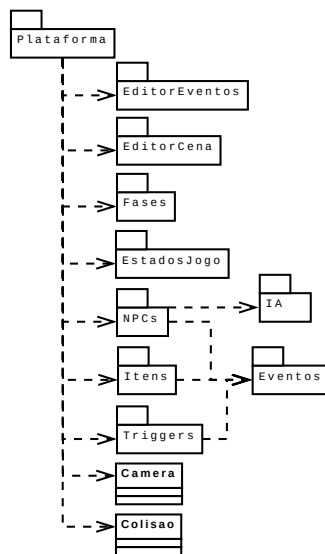


Figura 4: Pacotes.

Abaixo uma explicação de alguns pacotes:

- EditorEventos e EditorCena: pacotes que implementam as interfaces dos editores e as suas respectivas funcionalidades.
- Fases: pacote com as classes que herdam de Fase e implementam características específicas de cada fase do jogo (iluminação, lógica, etc.). O objetivo da plataforma é que estas classes sejam responsáveis também por carregar o modelo da fase; toda a sua construção será feita a partir dos editores integrados.
- EstadosJogo: classes que são utilizadas na máquina de estados geral da plataforma, como Carregando, Fim do jogo, Inventário, etc.
- IA: possui as classes que implementam aspectos envolvendo a inteligência artificial, como envio de mensagens entre entidades, algoritmos para movimentação de agentes, etc.
- Eventos: ações que poderão ocorrer em algum ponto do jogo, como alterar o estado de um NPC (*non-playable characters*), exibir um objeto, mudar de fase, etc.
- NPCs: pacote que implementa tipos de NPCs e estados específicos de cada um.
- Itens: classes que representam diferentes tipos de itens, como energia, arma, chave, objeto danoso, etc.
- Triggers: classes responsáveis por detectar a colisão entre objetos e gerar uma reação (evento) pré-definido.
- Camera: câmera em terceira pessoa.
- Colisao: *callback* que verificará as colisões entre os objetos.

2.1.1 NPCs

Os NPCs também são controlados por uma máquina de estados finitos, própria de cada NPC. Para a construção da plataforma proposta, alguns estados padrões já foram definidos para alguns tipos de NPCs, como mostrado na Figura 5.

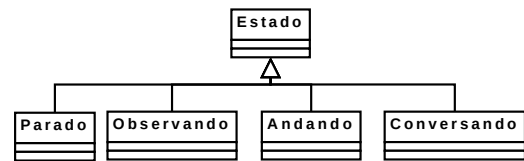


Figura 5: Classes que representam os estados de uma pessoa.

A definição de NPCs adicionais também é fácil com o uso do arcabouço, bastando estender da classe Pessoa, Inimigo ou NPC, como descrito na Figura 6. Cada novo NPC pode ter inúmeros estados e a transição entre eles é controlada sua pela máquina de estados definida na plataforma.

Alguns comportamentos de inteligência artificial já estão implementados, como perseguição, fuga, vagar, seguir caminhos (*way-points*), desvio de obstáculos. Cada objeto NPC possui um *orientador*, que é um apontador para um objeto que calcula o movimento do NPC a cada iteração do laço do jogo de acordo com os comportamentos ativados no momento (fuga, perseguição, desvio de obstáculos, etc) [Mat Buckland 2004].

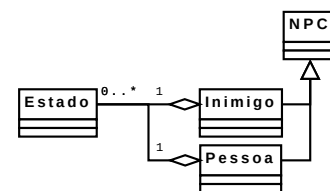


Figura 6: Estrutura de classes para um NPC.

2.1.2 Objetos do jogo

Os objetos visíveis na tela são do tipo das classes descritas na Figura 7.

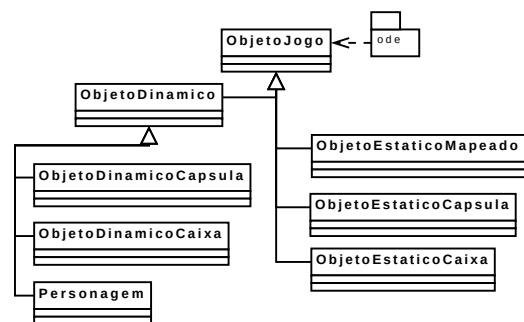


Figura 7: Classes que representam os objetos inseridos no jogo.

Objetos dinâmicos possuem um *corpo rígido* e uma *geometria de colisão*. O primeiro é responsável pelos aspectos cinemáticos do objeto, e o segundo é responsável por efetuar as colisões. Os objetos estáticos só possuem uma geometria.

Toda a construção das fases é feita instanciando diretamente estes objetos (ou algum de seus filhos), encapsulando assim alguns aspectos de menor nível de abstração.

2.2 Editores integrados

A construção das fases e eventos são facilitadas com os editores integrados à plataforma. Eles foram desenvolvidos como sendo novos estados da máquina de estados geral e por isso podem ser acessados durante a execução do próprio jogo.

Os dois editores utilizam arquivos XML para salvar informações, que são posteriormente interpretados no carregamento do jogo, com o auxílio da biblioteca *lxml*.

2.2.1 Editor de fases

O editor de fases é o passo inicial para a construção de uma fase. Com ele é possível inserir no cenário os seguintes objetos: Modelos (objetos que não possuem qualquer tipo de estado), *triggers*, itens, NPCs e caminhos (conjunto de pontos que poderá ser associado a um ou mais NPCs).

Sem o editor de fases, duas possíveis maneiras de se incluir objetos seriam: exportar o modelo do software de modelagem 3D já com a informação sobre a posição de cada objeto; ou então inserir a posição dos modelos diretamente no código. As duas alternativas não são práticas. A primeira pode criar um excessivo esforço em salvar e exportar os modelos e a segunda alternativa pode não ser atraente para quem não é familiarizado com programação.

Além disso, como o editor de fases já está integrado ao jogo, podemos observar os objetos imediatamente após a sua inserção e avaliar se ele deve ficar em outra posição ou se deve ter diferentes características.

Ao inserir os objetos, o usuário poderá alterar suas propriedades:

- Possui geometria: irá definir se o objeto possui ou não uma geometria, usada para detectar colisões.
- Elasticidade, Massa, Fricção: características do corpo e da geometria do objeto (caso possua).
- Tipo do objeto: estático (só possui uma geometria de colisão) ou dinâmico (possui, além da geometria, um corpo rígido e está suscetível às mudanças de posições decorrentes da cinemática).
- Forma da geometria: geometria de colisão que irá envolver o objeto. Pode ser um cilindro, uma esfera, uma cápsula.
- Tipo: definição do tipo do NPC (caso o objeto seja um), como por exemplo Pessoa ou Inimigo, o qual já possui alguns estados definidos, como citado anteriormente.
- Estado inicial, caso seja um NPC.
- Caminho: um NPC poderá ter um ou mais caminhos.

Uma tela do editor de fases é mostrada na Figura 8.

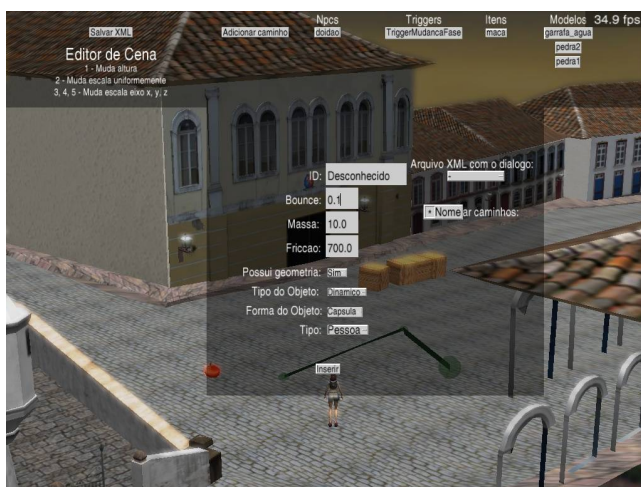


Figura 8: Tela do editor de fases.

Na Tabela 1 é exibido um arquivo XML como exemplo, onde há um personagem inserido na cena.

```

1 <cena>
2   <npc>
3     <id>vendedor</id>
4     <dialogos />
5     <tipo>Pessoa</tipo>
6     <estado>Parado</estado>
7     <modelo>
8       <arquivo>vendedor</arquivo>
9       <dirBase>personagens/humanos</dirBase>
10      <posicao>Point3(246.322, -160.502, 157.358)</posicao>
11      <escala>VBase3(0.0471012, 0.0471012, 0.0471012)</escala>
12      <rotacao>VBase3(0, 0, 0)</rotacao>
13      <visivel>Sim</visivel>
14    </modelo>
15    <geometria>
16      <geoTipo>Dinamico</geoTipo>
17      <geoForma>Capsula</geoForma>
18      <friccao>700.0</friccao>
19      <massa>10.0</massa>
20      <bounce>0.1</bounce>
21    </geometria>
22    <caminho>
23      <loop>Sim</loop>
24      <ponto>Point3(246.632, -159.704, 127.387)</ponto>
25      <ponto>Point3(272.94, -319.947, 117.257)</ponto>
26      <ponto>Point3(203.891, -318.641, 116.292)</ponto>
27    </caminho>
28  </npc>
29 </cena>

```

Tabela 1: Arquivo XML com a descrição de uma cena.

2.2.2 Editor de eventos

A inspiração para o uso de arquivos XML para o tratamento de eventos veio do que foi exposto em [Bastos et al. 2007]. Foi definida uma sintaxe própria e uma interface gráfica para a plataforma aqui proposta.

Este editor será o responsável por dar alguma coesão entre os objetos inseridos no editor de fases. O princípio dele é que qualquer *trigger*, *NPC*, *item* ou um próprio *evento* possam gerar um outro *evento*.

Como o relacionamento entre os objetos fica bastante claro na interface, as outras áreas envolvidas no desenvolvimento também podem usar o editor de eventos e, com isso, criar uma fase. O envolvimento do programador só seria necessário na criação dos eventos.

Uma série de eventos já foram definidos na plataforma:

- Exibir texto: exibe um texto (requer que um arquivo XML com a definição do texto seja selecionado).
- Conversa: inicia uma conversa (também requer que um arquivo XML com a definição do diálogo seja selecionado).
- Alterar fase: muda a fase em que o jogador se encontra.
- Alterar estado: altera o estado do objeto.
- Verifica vida: verifica a vida do jogador.
- Exibir/Ocultar objeto.

Para que o evento ocorra, é preciso que uma combinação de pré-requisitos sejam atendidos. Para facilitar a visualização, foi criada uma interface (Figura 9) em que é possível observar os *triggers*, NPCs e itens inseridos com o editor de fases e ainda inserir eventos ou verificadores e criar relacionamentos entre eles. Dessa forma, um evento só ocorrerá caso os seus pré-requisitos sejam satisfeitos.

Na implementação, cada objeto do tipo *Trigger*, NPC, Item ou Evento possui uma lista de apontadores para os seus pré-requisitos e cada um deles possui um valor booleano que descreve se ele foi ativado ou não. A ocorrência de um evento acontecerá caso todos os seus pré-requisitos estejam ativados.

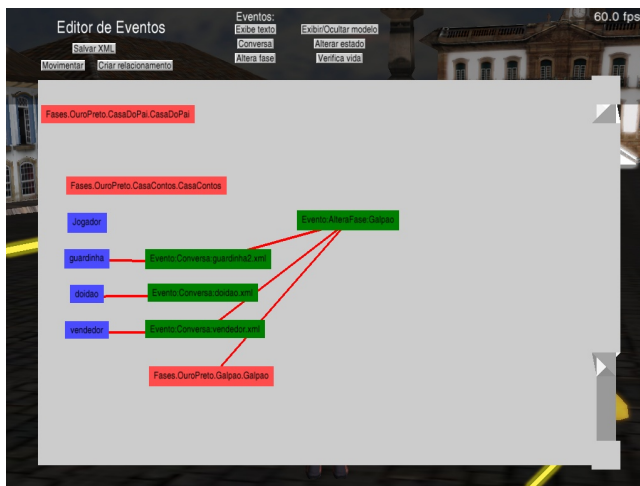


Figura 9: Tela do editor de eventos.

A Figura 9 mostra um exemplo simples de como o editor pode ser usado: o jogador só poderá mudar para a fase Galpão caso tenha ativado o *trigger* Galpão e conversado com os NPCs Vendedor e Guardinha. O arquivo XML salvo será como o exibido na Tabela 2.

```

1 <eventos>
2   <evento>
3     <tipo>Plataforma.Eventos.Conversa.Conversa</tipo>
4     <id>ConversaGuardinha</id>
5     <preRequisitos>
6       <npc>
7         <idRef>guardinha</idRef>
8       </npc>
9     </preRequisitos>
10    <xml>guardinha2.xml</xml>
11  </evento>
12  <evento>
13    <tipo>Plataforma.Eventos.Conversa.Conversa</tipo>
14    <id>ConversaVendedor</id>
15    <preRequisitos>
16      <npc>
17        <idRef>vendedor</idRef>
18      </npc>
19    </preRequisitos>
20    <xml>vendedor.xml</xml>
21  </evento>
22  <evento>
23    <tipo>Plataforma.Eventos.AlterarFase.AlterarFase</tipo>
24    <id>Galpao</id>
25    <preRequisitos>
26      <evento>
27        <idRef>ConversaVendedor</idRef>
28      </evento>
29      <evento>
30        <idRef>ConversaGuardinha</idRef>
31      </evento>
32      <trigger>
33        <idRef>Fases.Galpao.Galpao</idRef>
34      </trigger>
35    </preRequisitos>
36  </evento>
37 </eventos>

```

Tabela 2: Arquivo XML com a descrição dos eventos.

2.3 Fluxo de desenvolvimento

O funcionamento da plataforma e interação entre as equipes da construção de um jogo (arte, *game design* e programação) baseia-se no fluxo apresentado na Figura 10.

O roteiro e o *design* do jogo irão ditar quais deverão ser os modelos criados pela equipe de arte. Com os modelos prontos, estes são exportados para o formato .egg (carregado pelo Panda3D). Um modelo base (o cenário) é então carregado pela plataforma Pandorga, onde será possível inserir os outros modelos (como os personagens) através do Editor de Fases, e os eventos com o Editor de Eventos.

O roteiro e o *design* também irão determinar se algum novo estado deverá ser implementado para algum NPC, ou então algum

novo evento. Estas implementações poderão ser feitas em *Python*, de acordo com o que foi proposto no arcabouço.

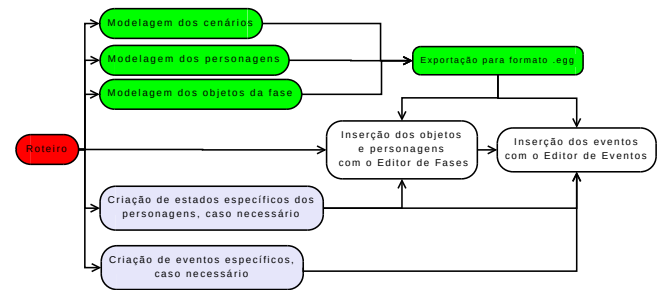


Figura 10: Fluxo de desenvolvimento.

3 Conclusão

Esta plataforma tem como principal objetivo elevar o nível de abstração necessário para a construção de um jogo. A proposta não busca substituir o papel do programador na criação, mas sim auxiliá-lo em algumas tarefas rotineiras na criação da base de um jogo e também permitir um contato mais direto de outras áreas, como a arte e o roteiro, com o jogo através dos editores. A Pandorga já se mostrou bastante útil no desenvolvimento do jogo Estrada Real Digital, diminuindo o atrito entre as equipes (principalmente entre a arte e a programação com o uso do editor de fases).

A plataforma ainda está em desenvolvimento, juntamente com o jogo e por isso está evoluindo de acordo com os requisitos deste. Para o futuro, planejamos integrar à plataforma componentes de áudio, como efeitos sonoros e músicas. Há planos também de um sistema para *Load* e *Save*, o que será facilitado pela atual representação em XML dos objetos do jogo.

Além disso, como o jogo ERD possui um aspecto educacional, a Pandorga poderia ser utilizada como um primeiro passo na educação de jovens quanto ao funcionamento de um jogo, aproveitando a interface amigável dos editores.

Agradecimentos

Gostaríamos de agradecer à FINEP, financiadora do projeto Estrada Real Digital, aos Professores Regina Helena Silva, Wagner Meira Jr., coordenadores do projeto, e também aos membros das equipes de arte (Carlos Augusto, André Persechini, Eduardo Fantini, Hamilton Alves, Wallison Ricardo) e roteiro (Juliana Franco, Raphael Almeida).

Referências

- BASTOS, A. S., MOURA, L. P., AND ALVES, L. R. G. 2007. Desenvolvimento de um sistema de roteiro para apoio ao processo de level design de um rpg educativo. *VI Brazilian Symposium on Computer Games and Digital Entertainment, 2007, São Leopoldo. SBGames 2007*.
- CORDEIRO, C. S., DE SOUSA, C. A. P., SAMPAIO, D. S., TEIXEIRA, L. G., DOMINGUES, B. F., TERRA, V. M., AND CHAIMOWICZ, L. 2006. Desenvolvimento de npcs para o jogo estrada real digital. *V Brazilian Symposium on Computer Games and Digital Entertainment, 2006, Recife. SBGames 2006*.
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISIDES, J. 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.
- GOSLIN, M., AND MINE, M. R. 2004. The panda3d graphics engine. *Computer* 37, 10 (Oct.), 112–114.
- MAT BUCKLAND. 2004. *Programming Game AI by Example*. Wordware Publishing, Inc.
- OLIVEIRA, A. R., CARDOSO, A. G., DOMINGUES, B. F., SAMPAIO, D. N., TEIXEIRA, L. G., MOREIRA, R. S., CHAIMOWICZ, L., FERREIRA, R., CARCERONI, R., AND JÚNIOR, W. M. 2005. Estrada real digital. *Anais do WJogos 2005 - IV Workshop Brasileiro de Jogos e Entretenimento Digital*, pp. 242–246. São Paulo, SP, Novembro de 2005.

A Framework for Genetic Algorithms in Games

Vinícius Godoy de Mendonça

Cesar Tadeu Pozzer

Roberto Tadeu Raittz

¹ Universidade Positivo, Departamento de Informática² Universidade Federal de Santa Maria, Departamento de Eletrônica e Computação³ Universidade Federal do Paraná, Curso de Tecnologia e Sistemas de Informação

Abstract

This article describes the architecture of a Genetic Algorithm Framework, techniques and optimizations used in its construction, a new selection algorithm called Asymptotic Selection, considerations about the use of this process in games and a usage example.

Keywords: artificial intelligence, genetic algorithms, biological programming, optimization techniques, evolutionary computation

Authors' contact:

vinigodoy@gmail.com
pozzer@inf.ufsm.br
raittz@ufpr.br

1. Introduction

Artificial Intelligence (AI) for Games often needs to deal with environments composed by several variables with unknown exact behavior. Consider a game like Sim City or Civilization, where the game AI advisor could suggest for the player an administrative strategy. This strategy should try to maximize several factors like the incoming money, production rate and technological advancements while reducing problems like pollution and famine. The consequence of each variable to the global scenario is not entirely predictable but, usually, given a set of defined values, it's easy to evaluate the whole picture.

Traditional AI offers several search mechanisms [Russel and Norvig 2003] which tries to find these answers. Some of them tries to systematically search the entire search space (like breadth-first search or A*). While they surely lead to the best possible solution, they may have a very long execution time. Another option would be using a local space search, like hill climbing search or Tabu search, but they tend to be locked in local maximums.

Genetic Algorithms (GA), are in the last category of the above two and try to use the principle of evolution, normally found in natural systems, to search for solutions to algorithmic problems [Schwab 2004].

The classical genetic algorithm defined by Holland [Holland 1975] is divided into six steps [Charles et al. 2008]: population creation, fitness evaluation,

chromosome mating and mutation, deletion of lowest fitness individuals and re-evaluation of the new population. The last steps are repeated until the convergence of the population to a good result.

One of the greatest advantages of GA is that it can be considered a parallel search mechanism that tests several different variables simultaneously [Charles et al. 2008]. Also, it avoids local maximums by doing mutations, and allowing searching of random areas of the solution space.

Unfortunately, this technique is not well known among the game developer community. There are few books on the subject and even less containing practical game situation examples. This article presents a framework that allows developers to easily integrate genetic algorithms into their games. The proposed solution is both flexible and easy to use.

The article also discusses some optimizations made in the framework. To improve understanding of the proposed architecture, a sample test scenario is provided in section 4.

2. Drawbacks of Genetic Algorithms

Time consuming evolution: Often evolution takes too many generations, even with a good genome designed with good operators [Schwab, 2004]. This is especially true in the presence of *deceptive problems* [Charles et al. 2008].

Evaluation by experimentation: Due to the great number of crossover, mutation, selection and scaling options, and due to the random nature of GAs, the only way to find a good solution is by experimentation [Schwab, 2004]. This also implies that a good genetic algorithm framework must ensure that experimenting is possible.

No guarantee of optimal solution: Just like any stochastic selection algorithm [Russel and Norvig 2003], there's no guarantee that the algorithm truly converged to the global maximum. On the other hand, for games finding good local maximum could be as good as choosing the global maximum especially because it could create a more human behavior [Schwab, 2004].

Hard to debug: Due to the random nature of the algorithm, it's hard to tell why a given implementation is not working.

3. Framework description

This section describes the entire framework organization and design considerations.

3.1. Domain specific classes

The first step when dealing with any genetic algorithm is to represent the problem in a form of a genome structure, and provide a fitness function to evaluate a specific individual.

In the proposed framework, there's a clear distinction between the concepts of an *Individual* and of a *Genome*. Such is not the case in several implementations like Schwab [2004], Charles et al. [2008] Jones [2008] and Obitko[1998]. So, the *Individual* has three very important roles:

1. To provide business methods for accessing domain specific properties;
2. To implement a common GA interface, allowing the framework to work;
3. To act as a *Mediator* [Gamma et al., 1995], for its genome structure and manipulation.

Notice that the individual models the business problem, so, it should be implemented by the framework user.

This approach has two major advantages:

1. The fitness function may use user-friendly methods to evaluate an individual, not manipulating the genome directly. This also makes the fitness function fully tolerant to genome structure changes;
2. The genetic framework does not depend upon a specific genome structure. User defined structures may be created and different implementations can be tested;

The fitness function is specified by an interface, and in case of C++ implementations a *functor* [Vandevoorde and Josuttis 2003] may be used instead.

3.2. Genome classes

The framework already provides the *BitGenome* class, that helps user's to model the problem as a bit string. This is the most common method [Schwab 2004 and Jones 2008]. *BitGenome* makes use of the *Strategy* design pattern [Gamma et al. 1995] for its crossover and mutation methods.

Two implementations of crossover are already provided: the point crossover method and a uniform crossover. Point crossover method provides a user-

defined number of crossover points, and could be used as a single or multi point crossover. For the mutation method, only the random mutation is provided.

The class diagram below describes this structure:

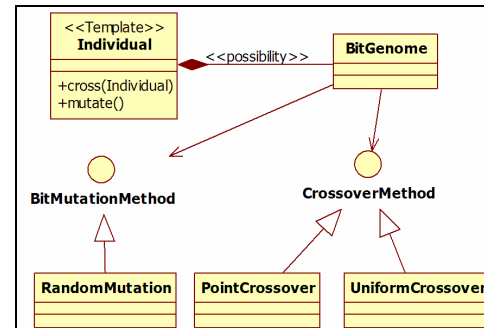


Figure 2: Bit genome –a possible individual internal structure

3.3. Scaling classes

Scaling is a common and optional technique to avoid the so called “super individuals” [Schwab 2004, Charles et al. 2008], that is, individuals with a score so high that they easily dominate the entire population. These may be a problem, since they may represent just a local maximum. The framework provides two common kinds of scaling: a rank scale and a sigma truncation scale [Schwab, 2004].

3.4. Selection function

The framework provides three selection methods proposed by Schwab [2004]: Tournament Selection, Roulette Wheel Selection and Stochastic Selection.

The traditional roulette wheel selection is described as follows [Obitko, 1998]:

```

S ← sum_of_all_fitness_scores(population)
r ← random(0, S)
for each individual in population
    s ← s + individual.fitness()
    if (s > r) then return individual
next

```

Figure 3: Traditional Roulette Wheel Selection

Exactly the same implementation is also found in Jones, 2008. This implementation assumes that the population is sorted in descending order.

The problem with this approach is that iterates through the population every time an individual must be selected. Since iteration is a time-consuming task, using this approach could be time killing for games. So an optimized implementation of the same algorithm was provided, allowing the iteration to occur only once:

```

S ← sum_of_all_fitness_scores(population)
for int i = 1 to desired_individuals_count do
    r[i] ← random(0,S)
next i
sort(r) //ascending order
lowest ← 0, sum ← 0, i ← 0
for individual in population
    s ← s + individual.fitness()
    while (r[lowest] < s)
        selected[i] ← individual
        i ← i + 1
        lowest ← lowest + 1
    wend
next
return selected

```

Figure 4: Optimized roulette wheel selection

3. The generation class also provides several defaults, based in the common options [Schwab, 2004]. These are: a crossover rate of 75%, an elitism rate of 5%, a mutation rate of 100% (but it considers that user will use the BitGenome, which default is 1% per bit), Roulette Wheel Selection and no scaling;

4. BitGenome fully implements the Individual interface, so it could be used directly. Off course, this will remove the benefits described in section 3.1, but allow a very simple usage of the framework;

4. Usage sample

To exemplify the usage of the framework, let's consider a Sim City like game where the player must manage the plantation of sugar maple (for biodiesel) and food. Player uses industry and food taxes to stimulate or decelerate each one of these plantation options.

Increasing the sugar maple fields will stimulate the industry, so the player will earn more money with taxes and the population will have more jobs. On the other hand, too many maple fields mean that the food will get more expensive, and it's quite possible that some people will get hungry. Increasing the food fields allow the player to make food cheaper, but it will slow down the industry. This could generate unemployment, and no working people will not buy any food or pay any taxes. They also generate other problems, such as crimes, not considered in our analysis. The exact impact of each parameter variation was configured by the game designer, in a script.

Now, let's consider an implementation of an advisor that suggests for the player a good approach about how to configure these taxes.

The first step is to configure an individual that contains all parameters that can be changed. They are the biodiesel industry tax and the food tax. We use the BitGenome structure to represent these taxes. The first

7 bits are used to store the industry tax and the last 7 bits to store the food tax.

After that, we will create the Fitness Function. The fitness function will use a game function that given the two taxes calculate the incoming money, rate of hungry people and rate of unemployed people. Notice that this function is commonly found in this kind of game, since it is also used for the game mechanics and to give support to the player decision.

The advisor will prefer individuals that contain:

1. No unemployment: People that don't work are hungry, do not contribute to the incoming taxes, and generate social problems, so they will be avoided to the maximum;

2. The biggest income as possible;

3. Little famine;

So, our fitness function could be:

```

unsigned calculate(Individual individual) {
    //First, discard invalid taxes
    if (individual.getIndustryTax() > 100)
        return 0;
    if (individual.getFoodTax() > 100)
        return 0;

    //Then calculate the projection
    Projection proj = calculateProjection(
        individual.getIndustryTax(),
        individual.getFoodTax());

    //Finally, calculate the score
    int score = proj.earnedMoney *
        1 - (proj.famineRate() +
        5 * proj.unemploymentRate());

    //Limit the lowest score to zero.
    return (unsigned) max(0, score);
}

```

Figure 5: Advisor fitness function

We can see that this advisor rejects unemployment five times more than famine. Also, earned money has a linear impact over the overall score and this impact is amortized by famine and unemployment.

Finally, let's include a sample of the "suggest" function. The function will use the genetic algorithm defaults. Taxes is our individual class. We will use an optional generation constructor that receives an *Abstract Factory* [Gamma et al. 1995] and generate *n* individuals using this factory.

```

Taxes Advisor::suggest() {
    //Create a generation
    Generation<Taxes> generation(factory, 30);

    //Calls next() many times, for 500ms
    generation.nextUntilTime(500);

    //Returns the best individual
    return generation.bestIndividual();
}

```

Figure 6: A sample advisor implementation

Even if a local maximum is returned, it has good chances to be a good advice. That's ok for this

implementation, since a human advisor is not perfect either, and could not see the best solution sometimes.

5. Conclusion

In this work, a genetic algorithm framework was presented, as well as its possible uses for games. Some optimization techniques were described, as well as several design considerations over the framework.

The asymptotic selection algorithm was presented, which gives advantages over speed when avoid scaling while using a simple implementation and easy of use.

The framework also presented a very flexible and customizable architecture, ideal for games. Most part of related works are suited for other areas like Galapagos (education) [AndrewMan, 2008] or Heat Exchangers [Tayal et. al 1998]. Similar work can be found for Java in JGAP [Meffert, 2008], but it does not leave a clear distinction in individual roles, like proposed here.

There are several options for future work, including increasing algorithm options in every step, introducing parallel processing and thread-safety to the framework or even creating self-adapting genome schemes.

The framework implementation is public and can be found in: <http://sofiaia.sourceforge.net>

References

- ANDREW MEN, GalapagosGA Framework. Available from: <http://sourceforge.net/projects/galapagosga> [accessed August, 2008]
- CHARLES, D., FYFE, C., LIVINGSTONE D., MCGLINCHIEY, S., Biologically Inspired Artificial Intelligence for Computer Games, *IGI Publishing*, 105-138.
- JONES, M. T., 2008. Artificial Intelligence: A Systems Approach. *Infinity Science Press Inc.*, 195-247.
- SMED, J., HAKONNEN, H., 2006. Algorithms and Networking for Computer Games. *John Willey and Sons*, 205.
- SCHWAB, B., 2004, AI Game Engine Programming, *Thomson Delmar Learning*, 413-452.
- NORVIG, P. AND RUSSEL, S.J., 2003. Artificial Intelligence: A Modern Approach, *Prentice Hall*, 116-119.
- VANDEVOORDE D., JOSUTTIS N. M., 2003. C++ Templates – The complete guide, *Addison Wesley*, 417-474
- OBITKO., MAREK. 1998. Introduction to Genetic Algorithms, [online], Hochschule für Technik und Wirtschaft Dresden. Available from: <http://obitko.com/tutorials/genetic-algorithms/> [accessed August, 2008].
- GAMMA E., RICHARD H. JOHNSON, R. VLISSIDES, J. 1995. Design Patterns – Elements of reusable object-oriented software, *Addison-Wesley Longman Inc.*
- HOLLAND, J. H., 1975. Adaptation in natural and artificial systems, *Ann Arbor MI, University of Michigan Press.*
- TAYAL, MANYSH C., FU YAN, DIWEKAR URMILLA, Optimal for Heat Exchangers: A genetic algorithm framework. <http://pubs.acs.org/cgi-bin/abstract.cgi/iecred/1999/38/i02/abs/ie980308n.html> [accessed August, 2008].
- MEFFERT, K., Java Generic Algorithm Platform (JGAP), Available from: <http://jgap.sourceforge.net/> [accessed August, 2008].

RPG-XEditor: um editor de jogos no estilo RPG baseado em XML

Rodrigo O. da Silva^{1,3}, Fábio A. C. Bossa^{1,3}, Rômulo C. Silva^{1,3}, Felipe H. Moreno²,
Eliane N. Pereira^{1,3}, Izaura M. Carelli^{1,3}, João A. Fabro³

¹ Universidade Estadual do Oeste do Paraná (UNIOESTE)

² Instituto de Tecnologia Aplicada e Inovação (ITAI)

³ Grupo de Pesquisa DETAE – Parque Tecnológico de Itaipu (PTI)
Foz do Iguaçu – PR – Brasil

Abstract

This paper reports the development of a RPG game editor, named RPG-XEditor, to create educational games. The traditional features of RPG are discussed to underlie how the RPG editor works. An overview of the software architecture RPG-XEditor is presented.

Palavras-chave: Editor de Jogos RPG, Jogos Educacionais, XML.

E-mail dos autores:

{felipe.h.moreno, fabiobossa.cc, imcarelli,
romulocesarsilva}@gmail.com,
rodrigo_okada@hotmail.com,
{joaofabro, enpbr}@yahoo.com.br

1. Introdução

Independentemente do tipo de jogo educacional a ser desenvolvido, as características de ludicidade e educação devem ser utilizadas como requisitos principais durante todo o processo de desenvolvimento do jogo. O professor pode participar ativamente deste processo, caso os desenvolvedores utilizem, por exemplo, o modelo de *design* participativo. No entanto, a liberdade de co-autoria do professor termina quando o desenvolvimento do jogo é finalizado. Em consequência disso, outros professores que desejarem utilizar o jogo deverão se adaptar ao conteúdo educacional previamente estabelecido nele. Além disso, outros fatores limitam a utilização de jogos educacionais, como por exemplo, seu custo e a quantidade reduzida de jogos existentes, não cobrindo os diversos conteúdos educacionais (História, Geografia, Ciências Naturais, Matemática, etc.).

As ferramentas RPG-XEditor e RPG-XPlayer foram desenvolvidas com o objetivo de apoiar o uso de jogos educacionais no estilo *Role Playing Game* (RPG). O RPG-XEditor permite a criação do roteiro do jogo, não exigindo conhecimentos técnicos de programação. Logo, professores de diversas áreas podem utilizá-lo para criar e atualizar seus próprios jogos educacionais. O RPG-XPlayer é o jogo propriamente dito, em sua versão atual ele executa RPGs com imagens 2D em perspectiva isométrica. A integração das duas ferramentas é feita através de arquivos XML (*eXtended Markup Language*) (ver figura 1). Ambas as ferramentas foram desenvolvidas utilizando a linguagem de programação Java e somente

softwares livres, sendo que o RPG-XPlayer utiliza a jMonkey Engine [Powell, 2007]. O objetivo foi reduzir o custo de desenvolvimento e disponibilizar ambas as ferramentas como softwares livres.

Este artigo tem por objetivo descrever o editor de jogos RPG-XEditor, mostrando como os elementos de um jogo RPG são criados, além de sua arquitetura de software.

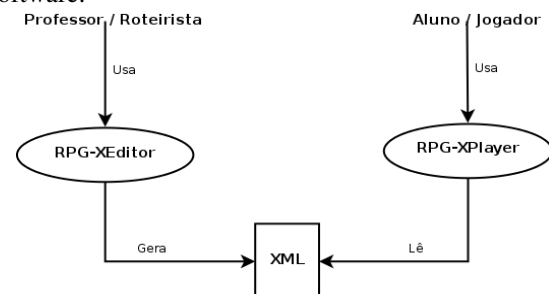


Figura 1: Integração entre RPG-XEditor e RPG-XPlayer

2. RPG

O RPG é um jogo de interpretação em que o jogador se movimenta pelo enredo do jogo, seguindo regras, superando obstáculos, cumprindo tarefas e geralmente aumentando as habilidades do seu personagem. O que difere esta modalidade de jogo das demais é que normalmente o enredo não tem fim [Salen e Zimmerman, 2004; Crawford, 1982]. Mesmo se o personagem morre, ele pode receber o atributo *vida* e continuar jogando.

No RPG há um personagem principal que interage com vários tipos de NPCs (*Non Player Character*), em um mundo definido por cenários, tendo que concluir missões (*quests*). O jogo possui um roteiro conciso sincronizado com o conteúdo proposto. Os RPGs eletrônicos tradicionais possuem dois tipos de NPCs: os que atacam (agressivos) o personagem, e os que não o fazem (não agressivos). Geralmente os NPCs não agressivos são utilizados para popular o cenário, mas há também os chamados *quest givers* que fornecem missões para o jogador.

As *quests* têm o papel de definir a interatividade do jogador com o roteiro do jogo. A maioria dos RPGs eletrônicos utilizam *quests* para expressar o que o jogador deve fazer para solucionar os conflitos, cumprindo assim missões que servem para dar emoção ao jogo e definir os objetivos que ele deve seguir no enredo do jogo.

O cenário do jogo é formado por um mapa com texturas que definem a área em que o personagem poderá interagir com os NPCs para cumprir as *quests* nas várias fases definidas no enredo do jogo. É o agrupamento desses componentes que define o clima do jogo. Um elemento essencial dos cenários são os itens que fazem parte da *quests* e que o personagem pode usar para cumprir as missões. Os itens são objetos presentes no jogo, com os quais o jogador interage para as realizações das missões definidas no roteiro do jogo.

3. Trabalhos Relacionados

Há alguns softwares *freeware* para a criação de jogos, como Game Maker, Click & Play, Stagecast Creator [Prensky, 2008] para desenvolver jogos que não demandam complexidades. As pesquisas de universidades no desenvolvimento de ferramentas para a criação de jogos já produziram editores de jogos para o ensino de programação para alunos da Ciência da Computação [Cooper et al., 2003]. Em 1995, o RPG Maker 95 foi o primeiro software para Windows que permitia que pessoas sem conhecimento técnico pudessem criar seus próprios jogos. A versão RPG Maker 2000 trouxe várias inovações e a possibilidade de criar jogos mais facilmente e rapidamente, mas esse programa só possibilita *quests* de combate.

No Brasil, “O que é, o que é?” [Pereira et al., 2008] é um ambiente de co-autoria de jogos educacionais contextualizados, com o apoio do conhecimento de senso comum, aplicado ao processo de aprendizado dos temas transversais propostos nos PCNs para o ensino fundamental. A co-autoria do professor nos jogos educacionais é um dos objetivos alvos do ambiente desenvolvido, agregado a essa característica está também a de se criar um ambiente que seja fácil para o professor utilizar e elaborar seus jogos de acordo com suas necessidades pedagógicas, sendo possível adaptar os jogos à realidade dos seus alunos.

Um dos diferenciais do RPG-XEditor é possibilitar mais interatividade, por exemplo o jogador receberá *quests* de conversa, coleta e sedex, além do tradicional combate, que possibilitará o jogador receber informação e interagir com o ambiente coletando material a ser entregue ao NPC.

4. RPG-XEditor

O editor de jogos RPG-XEditor (ver figura 2) é uma ferramenta, desenvolvida em Java 6.0, que permite especificar os requisitos necessários a criação de um RPG educacional, a saber: *itens*, *NPCs*, *missões*, *texturas*, e *cenários*. Todos esses elementos do RPG podem ser criados, alterados e excluídos através do editor, possibilitando ao professor/*game designer* personalizar o roteiro do jogo.

O RPG-XEditor pressupõe que o jogo a ser desenvolvido será utilizado pelo RPG-XPlayer. Um professor/*game designer* utiliza o editor para a criação do roteiro do jogo. Em seguida, o editor gera arquivos XML que serão lidos pelo RPG-XPlayer.

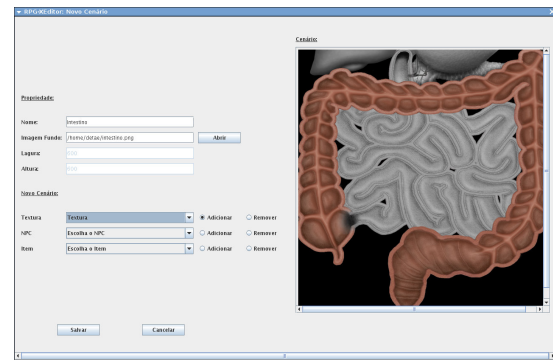


Figura 2: Tela de criação de Cenários do RPG-XEditor

4.1. Itens

Os itens são divididos em tipos: combate, defesa, consumo, coletável, portal e padrão. Um item de combate serve para causar danos a uma criatura presente no cenário, por exemplo, uma arma. Os danos podem ser: de perto ou à distância.

Os itens de defesa fornecem proteção ou funcionalidades ao personagem quando ativados, por exemplo, uma bota anti-ácida.

Um item de consumo é aquele que ao ser ativado realiza uma funcionalidade específica, sendo consumido ou extinto, por exemplo, uma poção de vida que aumenta em 10 pontos a vida do jogador.

Itens do tipo coletável permanecem distribuídos no cenário do jogo. Quando o jogador realiza um evento sobre ele (exemplo: clicar com o botão direito do mouse) é criado um item específico na mochila do jogador e o item original desaparece do cenário, podendo reaparecer posteriormente ou não, conforme as regras do jogo.

Um item do tipo portal permite a navegabilidade do jogador, efetuando a mudança de mapa.

Finalmente, um item padrão serve para criar o cenário, dentro do contexto do jogo, ou como componente de uma missão. Exemplos: fita de cabelo, cartaz ou pedra.

4.2. NPCs

Os NPCs são personagens com os quais o jogador poderá interagir. Eles são divididos em 5 tipos: *Quest Giver*, Amigo, Passivo, Agressivo e Neutro. O personagem *Quest Giver* é um fornecedor de missões, ele é responsável por pedir ao jogador para realizar uma determinada missão. O personagem Amigo não ataca o jogador, que também não pode atacá-lo. O personagem Passivo nunca irá atacar o jogador, mas o jogador pode atacá-lo. O personagem Agressivo é capaz de atacar o jogador, e o fará sempre que ele entrar na sua área de visão. Já o personagem Neutro somente ataca o jogador se este atacá-lo primeiro. É possível associar ao NPC diálogos, permitindo a interação do NPC com o jogador.

4.3. Missões

As *quests* são missões que devem ser cumpridas e que são disponibilizadas por um NPC *Quest Giver*.

Cada missão cumprida traz ou não o ganho de um item para o jogador. Existem 04 (quatro) tipos de missões: coleta, combate, conversa ou sedex.

Na missão do tipo coleta é requisitado ao jogador coletar uma quantidade determinada de itens. Esse tipo de missão tem por objetivo fazer com que o jogador explore determinado cenário do jogo.

Uma missão de combate está relacionada à eliminação de criaturas geralmente hostis ao jogador ou ao NPC que fornece a missão, e tem por objetivo destruir uma quantidade determinada de NPCs.

O objetivo de uma missão do tipo conversa é dar sequência à história que o jogador está vivenciando através da conversa com um determinado NPC.

Missões do tipo sedex têm por objetivo entregar um determinado item que seja importante para outro NPC, sendo outra forma de desenvolvimento da história.

É possível também escolher se uma missão terá ou não outra como pré-requisito, quais itens o jogador receberá para realizar a missão e também quais itens serão a recompensa do jogador.

4.4. Texturas

As texturas são pequenos componentes do solo, ou seja, é uma figura sobreposta à imagem original, podendo esta ser normal, bloqueadora ou hostil. A textura normal é sobreposta à imagem apenas para adicionar alguns detalhes que se repetem no cenário e não bloqueia o deslocamento dos personagens, enquanto a textura bloqueadora indica que a navegação não é possível. Ela pode estar associada a uma figura ou não, ou apenas bloquear uma parte do mapa. Já a textura hostil é aquela em que o jogador sofre um dano caso navegue sobre ela, podendo assim perder pontos de vida. As texturas são utilizadas na criação de cenários do jogo.

4.5. Cenários

A montagem de um cenário consiste em adicionar NPCs, itens e texturas a alguma imagem de fundo estática, que serve de mapa para o jogo. Após escolher a imagem ou figura que representa o mapa, o usuário seleciona o objeto e indica a posição que este deve ocupar dentro do cenário. Dessa forma, o usuário pode criar cenários de diferentes complexidades.

5. Arquitetura do RPG-XEditor

A Arquitetura do RPG-XEditor é composta por 03 (três) camadas baseada no conceito MVC (*Model-View-Controller*), ver figura 3.

A camada de Visão contém as classes que representam as telas, responsável pelo tratamento de eventos gerados pela interação do usuário com a interface do editor. Os eventos disparam chamadas de métodos das classes de Casos de Uso, localizadas na camada de Negócio, que utilizam o padrão de projeto Fachada (*Facade*) [Gamma et al. 1995].

As classes de Controle realizam chamadas às

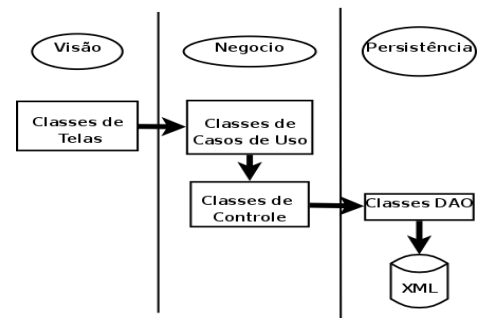


Figura 3: Camadas da arquitetura do RPG-XEditor

classes DAO (*Data Access Object*) na camada de Persistência, que fazem acesso aos arquivos XML utilizando a biblioteca JDOM (*Java Document Object Model*) [Hunter, 2004] do Java 6.0.

A Figura 4 representa um MER (Modelo Entidade-Relacionamento) entre os principais elementos que compõem o RPG-XPlayer. Cada entidade possui um arquivo XSD (*XML Schema Definition*) que descreve a estrutura do arquivo XML correspondente, gerado pelo RPG-XEditor.

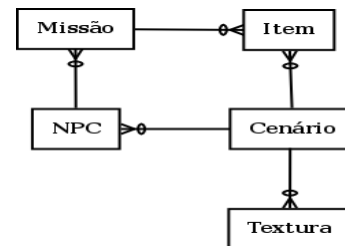


Figura 4: MER que representa a estrutura XML

6. ViaCorpo: um estudo de caso do uso do Editor de Jogos

O jogo *Viagem pelo Corpo Humano* (ViaCorpo), concebido no laboratório DETAE (Desenvolvimento de Tecnologias Aplicadas à Educação), tem por objetivo ser um jogo educacional estilo RPG que possibilite aos alunos revisarem os seus conhecimentos sobre o corpo humano na disciplina de Ciências Naturais de 7ª série do ensino fundamental, segundo os conteúdos pedagógicos descritos nos Parâmetros Curriculares Nacionais (PCNs).

O enredo do ViaCorpo é dividido em 7 (sete) fases, 06 (seis) delas correspondendo aos sistemas do corpo humano e 01 (uma) relacionado a vários deles, na seguinte ordem, a saber: sistemas Digestório, Respiratório, Circulatório, Excretor, Reprodutor, Nervoso, e Interação entre os Sistemas. Em cada uma dessas fases, o jogador é desafiado a explorar o respectivo sistema do corpo humano, através de missões que devem ser cumpridas, tais como combate a bactérias, coleta de nutrientes, entre outras.

O personagem principal do jogo, representando o próprio jogador, é um alienígena que veio à Terra para resgatar cientistas de seu planeta que vieram explorar o corpo humano. O alienígena é suficientemente pequeno para explorar os sistemas do corpo humano. A equipe

de designer ao definir o perfil do jogador optou por incluir a escolha do gênero do personagem: feminino e masculino. Ele inicia sua viagem pela boca, entrando assim pelo sistema Digestório.

O jogador interage com NPCs que representam criaturas presentes no corpo humano, tais como os cientistas alienígenas, bactérias e vírus. Alguns desses NPCs são do tipo *Quest Giver* (ex.: nave e cientistas alienígenas), outros são agressivos (ex.: a bactéria *Streptococcus mutans* que causa cáries nos dentes), outros são amigo (ex.: a bactéria *Bifidobacterium* presente no intestino grosso).

Algumas missões envolvem coleta de itens, tais como saliva, sucos digestivos, bile e alimentos. Na realização dessas missões é comum o uso de itens representando determinados instrumentos, tais como botas anti-ácida (item de defesa) e coletor de líquidos (item ativável).

Após terminar determinadas missões, o alienígena é automaticamente transportado para outro sistema do corpo humano, correspondendo à fase seguinte do jogo. Ele também tem a opção de retornar aos sistemas já visitados, usando para isso a interface de navegação de sua nave.

Os cenários do ViaCorpo correspondem a imagens dos órgãos dos diversos sistemas do corpo humano. (Ver Figura 5)

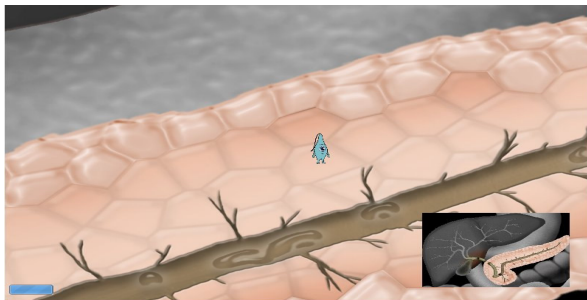


Figura 5. Exemplo de cenário com o personagem no pâncreas

6. Conclusões e trabalhos futuros

O *RPG-XEditor* é uma ferramenta para facilitar a criação de jogos RPG executados pelo *RPG-XPlayer*. A principal motivação para se criar o *RPG-XEditor* foi facilitar a criação de roteiros por parte do *game designer*, permitindo adicionar novos conteúdos, imagens mais atrativas, caso tenham se tornado “obsoletos”, ou haja necessidade de adaptá-los para outras séries ou faixas etárias. Tudo isso sem ser necessário conhecimento técnico de qualquer linguagem de programação.

O *RPG-XPlayer* foi desenvolvido como uma camada que especializa a *jMonkey Engine* [Powell, 2007] para jogos RPG. A utilização de arquivos XML para fazer sua integração com o *RPG-XEditor* permite que futuramente os jogos desenvolvidos no editor possam ser executados em outras *engines* que interpretem corretamente os formatos dos arquivos.

Todos os cenários pertencentes ao roteiro do jogo educacional *ViaCorpo* foram desenvolvidos utilizando

o *RPG-XEditor*, tornando o jogo mais flexível, pois o usuário poderá inserir novos conteúdos, tal como descrever o ciclo de uma doença específica, enriquecendo e tornando-o mais atrativo.

A versão atual do editor é *stand-alone*, utilizando a biblioteca *Swing* do *Java 6.0* na interface gráfica, e efetua a persistência das informações em arquivos *XML* diretamente.

Como trabalhos futuros pode-se incluir um banco de dados relacional (ex.: *MySQL* ou *PostgreSQL*) o qual permitiria que elementos de um jogo possam ser reaproveitados em outro, sendo os arquivos *XML* gerados a partir da base de dados. Além disso, a interface gráfica pode ser facilmente adaptada para utilizar a tecnologia de *applets*, habilitando a utilização do editor via Internet através de navegadores (*browsers*). Isto pode permitir a criação de um repositório de objetos de aprendizagem baseado em jogos RPG.

Pretende-se realizar um projeto piloto em escolas públicas do ensino fundamental para que professores de diferentes disciplinas (Geografia, História, etc.) possam avaliar a usabilidade do editor, assim como também, a avaliação dos alunos em relação ao jogo gerado por seus professores.

Agradecimentos

Os autores gostariam de agradecer à Financiadora de Estudos e Projetos (FINEP) pelo apoio financeiro, obtido através da Chamada Pública MCT/FINEP/MEC – Jogos Eletrônicos Educacionais– 02/2006. Gostariam também de agradecer todo o apoio do ITAI – Instituto de Tecnologia Aplicada e Inovação, e da empresa B3 Informática.

Referências

- CRAWFORD, C., 1982. *The Art of Game Design*. Disponível em: <http://www.vancouver.wsu.edu/fac/peabody/game-book/ACGD.pdf> [Acessado em 19 de agosto de 2008].
- COOPER, S.; DANN, W.; PAUSCH, R. *Teaching Objects-first In Introductory Computer Science*. In: SIGCSE 2003
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- HUNTER, J., 2004. *JDOM project*. Disponível em: <http://www.jdom.org/> [Acessado em 19 de agosto de 2008].
- PEREIRA, E. N.; FERREIRA, A. M.; ANACLETO, J. C.; CARVALHO, A. F. P.; CARELLI, I. M. *What is it?: A Culture Sensitive Educational Games* (prelo). In: 20th World Computer Congress, 2008, Milan.
- POWELL, M., 2007. *jMonkey Engine User's Manual*. Disponível em <http://www.jmonkeyengine.com/> [Acessado em 19 de agosto de 2008].
- PRENSKY, M. *Students as designers and creators of educational computer games: Who else?* In: British Journal of Educational Technology 2008 39:1 p. 1 -16.
- SALEN, K.; ZIMMERMAN, E., 2004. *Rules of Play: Game Design Fundamentals*. MIT Press, Cambridge, MA, U.S.A., 2003.

Criando níveis 3D para jogos móveis usando um editor 2D

Bruno M. Arôxa Luiz A. B. da Silva Artur F. Mittelbach

C.E.S.A.R, Brasil

Abstract

The main objective of this paper is to present a technique to ease the static building of 3D scenes for mobile games using the tile map editor, Mappy, as the level editor tool. The goals are to simplify the level edition using a 2D editing tool and to enable exporting directly to the game's engine. This approach cut out the need of a 3D level editor. Moreover, the technique enables the creation of low cost levels (in this case study 8KB per level and 27KB to the 3D repository).

Resumo

O objetivo deste artigo é apresentar uma técnica para facilitar a construção de cenários 3D estáticos em jogos móveis, utilizando Mappy Tile Map Editor – Mappy – como ferramenta de edição de níveis. A proposta é simplificar a edição de níveis utilizando uma ferramenta de edição 2D e exportá-lo diretamente para o motor do jogo. Essa abordagem elimina a necessidade de um editor de fases 3D. Além disso, a técnica permite a construção de níveis a um baixo custo (no estudo de caso deste trabalho 8kb por nível e 27kb do repositório de peças 3D).

Keywords: editor de níveis, jogos para dispositivos móveis, *level design*.

Authors' contact:

{bruno.aroxa, arturfm}@gmail.com
arturbotelho_4@yahoo.com.br

1. Introdução

A utilização de editores de fases (*level editor*) é o meio mais apropriado para o desenvolvimento de fases em jogos de computador. Isso porque permite ao *designer* montar as suas fases, visualizando-as em um mundo próximo ao que será visto no jogo, assim como permite a edição ágil de qualquer mudança que possa vir a ser feita.

Entretanto, dependendo do tipo do jogo abordado, podemos usar uma abstração e representar níveis 3D através de uma representação 2D. Apesar de o projetista ter que abrir mão da visualização completa do nível, ele ganha uma forma de edição na qual poderá capturar mais facilmente as informações necessárias para o motor do jogo recriar o cenário durante a sua execução.

O objeto de pesquisa deste documento ajuda a resolver alguns problemas referentes à facilidade e

rapidez na hora da criação e edição dos níveis do jogo. O Mappy [2008] oferece um ambiente com visão 2D simples e de fácil entendimento, que não requer conhecimento prévio de ferramentas 3D.

A técnica proposta aqui é baseada na geração de arquivos binários de nível onde estão presentes as informações necessárias para que o motor crie o mundo usando peças 3D e texturas presentes num repositório criado para o jogo. O uso dos arquivos citados proporciona uma excelente economia de espaço de memória do dispositivo. Este trabalho utilizará como estudo de caso um jogo 3D de corrida de carros, rodando em um dispositivo sobre a plataforma Java (CLDC [2008] 1.1, MIDP [2008] 2.0 com suporte à API M3G [2008]). Apesar de o estudo de caso ser realizado com um jogo de ambiente aberto, a técnica aqui proposta também pode ser aplicada à jogos de ambientes fechados que possam ser projetados em um plano 2D.

Este artigo está estruturado da seguinte forma: a seção 2 cita outros trabalhos relacionados com o tema proposto; a seção 3 apresenta as ferramentas envolvidas na construção dos níveis; a seção 4 detalha as etapas do processo de criação dos níveis; finalmente, a seção 5 expõe as conclusões e considerações finais deste trabalho.

2. Trabalhos Relacionados

Até o momento os autores não encontraram nenhuma referência para trabalhos relacionados ao tema proposto. No entanto, outros trabalhos relacionados a editores de níveis foram apresentados, entre eles destacam-se: a proposta de um *framework* para a criação de editores de níveis [Filho et al. 2004]; e um trabalho relacionado ao uso de editores de níveis para a construção de cenários para jogos 3D, que foi proposto por Salvi et al. [2006]. Este último se diferencia no fato de utilizar um editor 3D e não 2D, como propõe este trabalho.

3. Ferramentas Utilizadas

A seguir serão descritas de forma sucinta as ferramentas relacionadas com o processo de criação de cenários e de *level design*, utilizadas neste projeto.

3.1 3DS Max

O 3DS Max [2008] foi usado nesse projeto para criar o repositório de peças que seria utilizado para a montagem dos níveis. O 3DS Max foi escolhido como

ferramenta para criação dos níveis por quatro motivos básicos:

1. O projeto já possuía licença do software.
2. O time já tinha know-how da ferramenta.
3. O time já tinha know-how dos exportadores 3DS – M3G [2008].
4. Vasta biblioteca de material de pesquisa.

3.2 Mappy Tile Map Editor

Foram realizados testes com dois editores gratuitos: Mappy, e o Tile Studio [2008]. Apesar de se apresentar como um produto aparentemente mais elaborado e estável, o Tile Studio não contava com todas as funcionalidades necessárias para suprir as necessidades do jogo. O Mappy foi escolhido, entre outros fatores, por permitir a edição de níveis em multicamada.

3.3 Lua Script

Lua [2008] é uma linguagem de *script* interpretada utilizada para expandir aplicações de forma geral. Ela é usada como forma de linguagem extensível que une partes de um programa feitas em mais de uma linguagem.

O fato de Lua ser uma linguagem de *script* suportada pelo Mappy, e apresentar características como portabilidade e simplicidade, direcionou a sua adoção no projeto.

4. Processo de criação dos níveis

O processo de criação dos níveis no contexto deste projeto seguiu etapas bem definidas, como é apresentado na Figura 1.

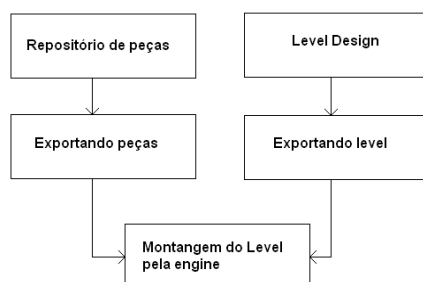


Figura 1: Etapas do processo de criação do nível do jogo.

As próximas subseções apresentam em detalhes cada fase do processo de criação e montagem dos níveis no jogo.

4.1 Definindo o Repositório de Peças

O artista 3D, com o apoio do projetista do nível, define os elementos necessários para compor o cenário do jogo. Por se tratar de um jogo de corrida ambientado num cenário urbano, vários dos elementos componentes de cenário são prédios e poderiam ser

tratados apenas com faces retangulares com aplicações de textura adequadas.

Grande parte do trabalho, entretanto, foi chegar ao conjunto mínimo de peças e formas de posicionamento para gerar o sentimento adequado de ambiente urbano. Durante esta fase foi necessário projetar e testar várias configurações de elementos gráficos e visualização para garantir a melhor configuração para o jogo. Entre elas:

1. *Ângulo das câmeras*: Garantindo que o jogador poderia ter visão ampla para o mundo do jogo e informação suficiente para tomar decisões relevantes para o jogo.
2. *Velocidade aparente*: Os primeiros testes com o jogo não apresentaram a sensação de velocidade esperada, apesar de os componentes da velocidade estar fisicamente coerentes. Para melhorar essa sensação foram feitos ajustes de câmera e aplicados efeitos gráficos tanto nos elementos da pista quanto nas texturas do mundo.
3. *Perspectivas e pontos de fuga*: Ajustes no posicionamento dos objetos da cena para proporcionar uma melhor visualização.
4. *Nível de detalhe das texturas*: Foram necessárias várias iterações para identificar o nível ideal de detalhe das texturas, de forma a obter o melhor custo-benefício para o jogo.

Após vários testes conseguiu-se reduzir o conjunto de peças do jogo para quatro peças:

1. Blocos de pista: peças que compõem as rodovias;
2. Limites (*guardrails*): peças que limitavam o movimento do jogador separando a área de jogo do cenário;
3. Cenário: composto basicamente por planos retangulares simples justapostos, com texturas diversas aplicadas: prédios, cercas gramadas, casas, lojas, delegacias, etc.;
4. Elementos especiais: Faixas de largada, placas de sinalização, pontos de som 3D, etc.

4.2 Exportando o Repositório de Peças

O repositório exportado (arquivo ‘.m3g’) contém o conjunto de peças utilizadas para a montagem dinâmica das pistas do jogo. Para a visualização desses arquivos foi utilizada a ferramenta M3G *Viewer*, que faz parte do pacote de ferramentas disponibilizado pela Mascot Capsule [2008].

Um dos maiores benefícios trazidos por este visualizador é a possibilidade da validação dos identificadores das geometrias antes mesmo da carga do arquivo na aplicação. Problemas no processo de exportação eram identificados com antecedência, o que tornou o trabalho mais produtivo.

4.3 Definindo o Level Design

O projetista do nível elabora os conceitos de cada um dos níveis já prevendo as devidas curvas de dificuldades e os sucessivos desafios do jogo. Com esses elementos em mãos, foram usadas diferentes camadas no Mappy para representar a sobreposição de alguns elementos, por exemplo, numa mesma área do mapa era possível representar uma pista, uma faixa de largada, uma placa de sinalização e uma parede de concreto como limitador. Foi necessário acostumar-se com a abstração e passar a usar símbolos comuns entre as camadas para representar elementos diferentes no jogo. A Figura 2 apresenta um exemplo de nível. No painel à direita, são apresentados os tipos de peças disponíveis para a construção.

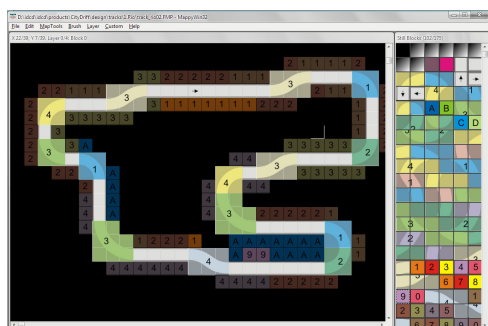


Figura 2: Edição de nível, tiles mapeados indicando cada peça e propriedade dos cenários.

Cada bloco no painel de peças possui um índice que o representa na matriz do nível. Uma vez definidos os parâmetros e necessidades básicas, esses índices foram mapeados manualmente para os tipos de peças modeladas no repositório.

Para este jogo, foram definidas as seguintes camadas:

1. *Camada de pista*: indicando qual peça do repositório seria utilizada para traçar a pista a ser corrida;
2. *Prédios (forma)*: usado para indicar a sucessão de planos de prédios fora da área de pista que simularia o ambiente urbano;
3. *Prédios (textura)*: uma vez definidos a posição e tamanhos dos planos/prédios foi possível referenciar a textura a ser aplicada neles;
4. *Camada de limites da pista*: indicando que tipo de separador seria usado entre a pista e o cenário: grades, canteiros de concreto, cercas vivas, etc.;
5. *Camada de som 3D*: indicando os pontos onde seriam aplicadas fontes de som 3D.

4.4 Exportando o Nível

Um *script* escrito em Lua foi utilizado em conjunto o Mappy, para gerar o arquivo binário contendo as informações necessárias para criar as pistas do jogo. Após finalizar a edição do nível, é realizado o processo de exportação onde é gerado um arquivo binário para a construção do cenário.

A representação visual do nível é interpretada pelo exportador como um conjunto de matrizes, onde cada camada do nível possui o conjunto de identificadores que serão utilizados pelo motor de montagem, conforme ilustra a Figura 3.

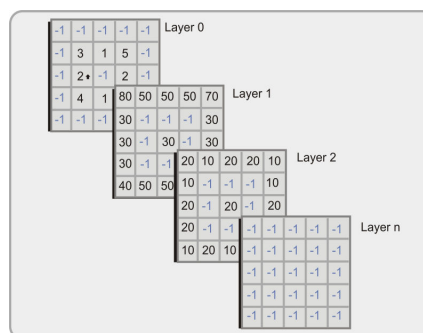


Figura 3: Estrutura de matrizes do nível.

Todas as camadas possuem as mesmas dimensões, e o identificador '-1' indica uma posição vazia na matriz. O arquivo binário está organizado da seguinte forma:

1. *Número total de camadas*: Representa o número total de camadas contidas no nível;
2. *Dimensão da matriz*: Define a largura e altura de todas as matrizes;
3. *Informações das matrizes*: Contém o conjunto de identificadores presentes na matriz da camada correspondente.

4.5 Carregando o Nível

A última etapa do processo de montagem do cenário consiste no mecanismo de carga do nível no motor de renderização 3D do jogo. Essa carga dos blocos que compõe cada pista é feita através do conjunto de classes apresentadas na Figura 4.

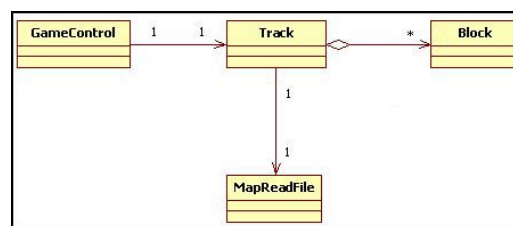


Figura 4: Representação UML das classes envolvidas no processo de carga do cenário no jogo.

A seguir são descritas as principais responsabilidades de cada classe:

- *GameControl*: Gerencia o fluxo de informação do jogo, delegando atividades para as devidas classes responsáveis.
- *Track*: Classe responsável por montar o nível. Inicialmente carrega as informações do nível através da classe *MapReadFile* e vai criando

novos blocos (instância de *Block*), que serão adicionados ao cenário do jogo.

- *Block*: Classe responsável por construir os blocos e elementos 3D que compõem o circuito do jogo.
- *MapReadFile*: Classe responsável pela leitura das informações contidas nos arquivos binários que são gerados pelo script.

O algoritmo de montagem faz uma busca na matriz, que é fornecida pela classe *MapReadFile*, para encontrar o identificador e os índices (x, y) do primeiro bloco a ser inserido no mundo 3D. O identificador encontrado é único e é o ponto de partida para inserir os blocos seguintes. Com o índice do primeiro bloco gravado, o algoritmo percorre a matriz em busca de novos identificadores, sempre analisando as posições vizinhas ao índice atual e, quando encontra um valor válido, cria o bloco que representa essa informação através da classe *Block* e insere no nível.

A inserção dos blocos no nível é feita com base no identificador da peça e na posição do bloco, de forma a garantir que cada bloco só será inserido no mundo uma única vez. A Figura 5 apresenta uma representação de como a peça é adicionada ao mundo.

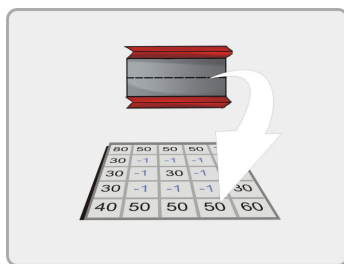


Figura 5: Representação de uma peça na matriz de nível.

5. Conclusão

A utilização do Mappy para abstrair a edição de nível de um jogo 3D para um ambiente 2D, reduziu a complexidade da criação e manutenção dos níveis, e se mostrou bastante eficaz para este trabalho.

Além disso, observou-se também uma redução expressiva do tamanho do jogo. Com a abordagem do repositório de blocos para a montagem dos níveis, foi possível oferecer uma grande quantidade (24 pistas) de níveis no jogo a um baixo custo (cerca de 8kb por nível e 27kb do repositório). Para o contexto do projeto, esse foi o maior benefício obtido neste trabalho.

Contudo, alguns problemas foram enfrentados durante a execução do projeto, expondo algumas limitações do modelo proposto. Entre essas limitações pode-se destacar o problema com o espelhamento de peças que, por uma limitação do motor gráfico, exigiu a duplicação de algumas peças; e o fato de só ser possível visualizar o resultado final do nível após a sua carga completa no jogo.

Outra limitação diz respeito ao alto acoplamento do motor de montagem ao domínio do jogo, requerendo um alto esforço para a adição de novas peças para compor os níveis, inviabilizando a sua utilização para outros domínios de jogos. Segundo Filho et al. [2004], esse alto acoplamento é advindo das características intrínsecas de cada jogo (características, regras, funcionalidades, etc.). Neste caso, uma melhoria clara para trabalhos futuros seria uma refatoração nas classes que compõem o módulo de carga, de forma a facilitar a inserção de novos tipos de peças.

Agradecimentos

Os autores agradecem ao C.E.S.A.R [2008] pela excelente infra-estrutura provida para a execução e sucesso deste projeto.

Referências

- 3DS MAX, 2008. Disponível em: <http://usa.autodesk.com/adsk/servlet/index?id=5659302&siteID=123112> [Acessado em 14 Ago. 2008].
- C.E.S.A.R, 2008. Disponível em: <http://www.cesar.org.br> [Acessado em 12 Ago. 2008].
- CLDC – CONNECTED LIMITED DEVICE CONFIGURATION API. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html> [Acessado em: 03 Out. 2008].
- FILHO, W. M. A., RAMALHO, G. L. E FRANCA, A. C. 2004. GEF – Um Framework para Editores de Níveis. In: *III Workshop Brasileiro de Jogos e Entretenimento Digital* parte do *SBGames 2004*, 20-21 Out. Curitiba.
- LUA, 2008. Disponível em: <http://www.lua.org/> [Acessado em 25 Ago. 2008].
- M3G – MOBILE 3D GRAPHICS API. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr184/index.html> [Acessado em: 25 Ago. 2008].
- MAPPY EDITOR, 2008. Disponível em: <http://www.tilemap.co.uk/mappy.php> [Acessado em 25 Ago. 2008].
- MASCOT CAPSULE M3G TOOLKIT, 2008. Disponível em: <http://www.mascotcapsule.com/toolkit/m3g/en/index.php> [Acessado em: 25 Ago. 2008].
- MIDP – MOBILE INFORMATION DEVICE PROFILE API, 2008. Disponível em: <http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html> [Acessado em: 03 Out. 2008].
- TILE STUDIO, 2008. Disponível em: <http://tilestudio.sourceforge.net/> [Acessado em: 25 Ago. 2008].
- SALVI, J. L., SANTOS, M. C., TORTELLI, D. M. E BRANCHER, J. D. 2006. Utilizando o 3D Studio Max como Level Editor para Construção de Cenários para Ogre3D. In: *V Workshop Brasileiro de Jogos e Entretenimento Digital* parte do *SBGames 2006*, 8-10 Nov. Recife.

Uma Comparação Preliminar entre Tecnologias para Implementação de Variabilidades em Jogos para Celulares

Rogério Celestino dos Santos, Marco Túlio Valente
Instituto de Informática, PUC Minas
rogerio.celestino@gmail.com, mtov@pucminas.br

Resumo

Descreve-se e compara-se neste artigo um conjunto de tecnologias para implementação de variabilidades na área de jogos para celulares. Para isso, mostra-se um estudo de caso onde uma variabilidade de um jogo para celular foi implementada por meio de três tecnologias: compilação condicional, programação orientada por aspectos e programação orientada por *features*.

Keywords: Linhas de produtos de software, Programação orientada por *features*, Programação orientada por aspectos, Compilação condicional, Jogos para celulares

1 Introdução

O desenvolvimento de jogos para dispositivos móveis é mais simples se comparado com o desenvolvimento para PCs e consoles. No entanto, jogos para dispositivos móveis devem ser adaptados aos diversos tipos de aparelhos celulares existentes. Normalmente, isso requer a manutenção de versões desses sistemas para diferentes plataformas de celulares, de forma a lidar com características particulares dessas plataformas, incluindo APIs de desenvolvimento proprietárias e restrições de *hardware* (tais como tamanho do *display*, quantidade de memória, diferentes interfaces de rede etc).

Assim, jogos para celulares constituem um domínio de aplicação promissor para desenvolvimento baseado em linhas de produto de software (LPS). Basicamente, essa abordagem de desenvolvimento propõe a derivação sistemática de produtos de software a partir de um conjunto de componentes e artefatos comuns [Clements and Northrop 2001]. Para tanto, engenheiros de software devem procurar identificar ao longo de todo processo de desenvolvimento pontos de variabilidade no núcleo de componentes e artefatos comuns, a partir dos quais possam ser derivados produtos específicos. No caso específico de jogos para celulares, essas variabilidades podem incluir o uso de uma API de desenvolvimento proprietária ou então *features* que devem existir em apenas algumas versões do sistema, como explosões ou animações mais complexas.

Diversas tecnologias têm sido propostas para implementação de variabilidades em LPS, incluindo compilação condicional (CC), programação orientada por aspectos (AOP) [Kiczales et al. 1997] e programação orientada por *features* (FOP) [Batory et al. 2003]. Particularmente, neste artigo, descreve-se uma experiência de utilização dessas três tecnologias na modularização de uma *feature* tradicional em jogos para celulares: efeitos sonoros. Para isso, foi utilizado como estudo de caso uma implementação de código livre em J2ME do jogo Chuckie Egg¹. Chuckie Egg é um jogo clássico da década de 80 cujo objetivo é fazer com que o personagem principal recolha a maior quantidade possível de ovos, desviando-se de galinhas que ficam protegendo os mesmos. Uma tela do jogo pode ser vista na Figura 1. A versão considerada do jogo Chuckie Egg possui 1519 linhas de código Java/J2ME e 15 classes.

Apresenta-se também no artigo uma avaliação qualitativa preliminar dos benefícios e desvantagens de cada uma das três tecnologias de implementação de variabilidades utilizadas no estudo de caso. O objetivo final da experiência e da avaliação apresentadas é subsidiar desenvolvedores de jogos que estejam interessados em utilizar em seus sistemas tecnologias mais modernas para implementação e modularização de variabilidades, como orientação por aspectos e orientação por *features*.

O restante deste artigo está organizado conforme descrito a seguir.

Na Seção 2, descreve-se brevemente cada uma das três tecnologias de implementação de variabilidades adotadas no artigo (CC, AOP e FOP). Na Seção 3, descreve-se a implementação de uma *feature* responsável pela introdução de efeitos sonoros no jogo Chuckie Egg usando essas três tecnologias. A Seção 4 apresenta uma avaliação das tecnologias utilizadas no estudo de caso. A Seção 5 apresenta brevemente trabalhos relacionados e a Seção 6 conclui.



Figura 1: Tela do jogo Chuckie Egg

2 Tecnologias para Implementação de Variabilidades

Descreve-se resumidamente nesta seção as três tecnologias para implementação de variabilidades consideradas neste artigo.

Compilação condicional (CC) é um mecanismo de implementação de variabilidades largamente utilizado em linguagens como C e C++. Basicamente, diretivas de pré-processamento são usadas para delimitar linhas do código fonte que devem ser incluídas (ou não) em uma determinada versão de um sistema. Em Java, não existe suporte nativo a compilação condicional. No entanto, existem ferramentas de terceiros que dão suporte a essa técnica de implementação de variabilidades, como, por exemplo, a ferramenta Antenna². Essa ferramenta utiliza um símbolo especial (`//#`) para indicar diretivas de compilação. A Figura 2 exemplifica a utilização dessas diretivas para delimitar o código responsável pela exibição de mensagens em um determinado sistema. Quando escolhida a diretiva `Portuguese`, a linha 2 será compilada. Por outro lado, a linha 4 somente será compilada quando a diretiva `English` for ativada em tempo de compilação.

```
1: // #if Portuguese
2: // @ public String Die="Morreu...";
3: // #elif English
4: // @ public String Die="Die...";
5: // #endif
```

Figura 2: Exemplo de Compilação Condicional

Programação orientada por aspectos (AOP) é uma técnica para separação de interesses transversais presentes no desenvolvimento de sistemas [Kiczales et al. 1997]. Interesses transversais são interesses que estão espalhados e entrelaçados em diversos módulos de um sistema. Dentre as linguagens com suporte a AOP, AspectJ é atualmente aquela mais madura e estável [Kiczales et al.

¹Disponível em <http://www.morgadinho.org/chuckie/>

²Disponível em <http://antenna.sourceforge.net/>

2001]. Basicamente, AspectJ oferece os seguintes recursos para modularização de interesses transversais: *join points* (pontos da execução de um sistema orientado por objetos que podem ser instrumentados por meio de aspectos), *advices* (trechos de código executados em determinados *join points*) e aspectos, os quais são semelhantes a classes, exceto pelo fato de possuírem como membros designadores de conjuntos pontos de junção e *advices*. AspectJ permite ainda modificar as assinaturas estáticas das classes e interfaces de um programa Java. Por meio de aspectos pode-se, por exemplo, introduzir novos campos e métodos nas classes de um sistema.

Programação orientada por features (FOP) é uma outra técnica de modularização e separação de interesses, particularmente adequada para implementação de LPS [Batory et al. 2003]. Basicamente, na terminologia de FOP, uma *feature* representa um acréscimo na funcionalidade básica de um sistema. Ou seja, FOP defende que sistemas devem ser sistematicamente construídos por meio da definição e composição de *features*, as quais são usadas para distinguir os sistemas de uma mesma família de produtos. AHEAD (*Algebraic Hierarchical Equations for Application Design*) é um conjunto de ferramentas que implementam os conceitos básicos de FOP [Batory 2004]. O principal componente desse ambiente é uma extensão de Java, chamada Jakarta (ou simplesmente Jak), que permite a implementação de *features* em unidades sintaticamente independentes. Por meio dessas unidades, chamadas de refinamentos, pode-se adicionar novos campos e métodos em classes de um programa orientado por objetos. Pode-se ainda adicionar comportamento extra em métodos dessas classes.

3 Estudo de Caso

A versão original do jogo Chuckie Egg não possui suporte a efeitos sonoros. Assim, para realização da comparação descrita neste artigo, resolveu-se incorporar essa *feature* no sistema usando cada uma das três tecnologias para implementação de variabilidades descritas na Seção 2. Basicamente, foram inseridos sons com músicas de fundo durante todo o jogo e efeitos sonoros quando o personagem principal pula sobre uma das galinhas e quando ele captura algum objeto.

Por limitações de espaço, uma única classe do sistema será usada no restante desta seção para ilustrar as implementações realizadas. A classe escolhida, chamada *Chuckie*, é responsável pela manipulação do personagem principal, incluindo a implementação de ações como pular, andar, subir escadas etc. Nesta classe, foi introduzido um efeito sonoro toda vez que o personagem principal executa um pulo.

Um fragmento do código da classe *Chuckie* é apresentado na Figura 3. Essa classe possui métodos que tratam os pulos do personagem principal, incluindo pulos em que ele permanece parado (linhas 6-8), pulos em que ele se move para a direita (linhas 9-11) e pulos em que ele se move para a esquerda (linhas 12-14). Utilizando as três tecnologias para implementação de variabilidades tratadas no artigo, descreve-se nas subseções seguintes como esses métodos podem ser instrumentados para incorporar a *feature* de som proposta no estudo de caso.

```

1: public class Chuckie extends GameSprite {
2:     .....
3:     public Chuckie(Image img, int x, int y) {
4:         .....
5:     }
6:     public void jump() {
7:         .....
8:     }
9:     public void jump_right() {
10:        .....
11:    }
12:    public void jump_left() {
13:        .....
14:    }
15: }

```

Figura 3: Fragmento de código da classe *Chuckie*

3.1 Compilação Condicional

Na Figura 4, apresenta-se a versão da classe *Chuckie* que utiliza CC para implementação da *feature* de Som. Inicialmente, utiliza-se uma diretiva de compilação para declarar um atributo do tipo *PlaySounds*, o qual faz o tratamento de som (linhas 2-4). Esse atributo é inicializado nas linhas finais do construtor da classe (linhas 8-10). Introduce-se também na classe um método que executa o som de pulo (linhas 13-21). Esse método foi criado para evitar repetições de código. Assim, nos métodos que implementam o pulo do personagem principal foram apenas inseridas chamadas ao método introduzido (linhas 24-26, 30-32 e 36-38). O som será executado logo no início do corpo desses métodos.

Assim, caso a *feature* Sound e sua respectiva diretiva de compilação sejam habilitadas, será gerada uma versão do sistema em que um efeito sonoro ocorre toda vez que o personagem principal realizar um pulo. Como pode ser observado, utilizando diretivas de compilação condicional, o código da *feature* fica entrelaçado no código base, aumentando seu tamanho e, portanto, dificultando o entendimento.

```

1: public class Chuckie extends GameSprite {
2:     // #if Sound
3:     // @ protected PlaySounds playJump;
4:     // #endif
5:     .....
6:     public Chuckie(...) {
7:         .....
8:         // #if Sound
9:         // @ playJump = new PlaySounds(...);
10:        // #endif
11:    }
12:    .....
13:    // #if Sound
14:    // @ protected void playJump() {
15:    //     try {
16:    //         playJump.play();
17:    //     } catch (Exception e) {
18:    //         .....
19:    //     }
20:    // @ }
21:    // #endif
22:    .....
23:    public void jump() {
24:        // #if Sound
25:        // @ playJump();
26:        // #endif
27:        .....
28:    }
29:    public void jump_right() {
30:        // #if Sound
31:        // @ playJump();
32:        // #endif
33:        .....
34:    }
35:    public void jump_left() {
36:        // #if Sound
37:        // @ playJump();
38:        // #endif
39:        .....
40:    }
41:    .....
42: }

```

Figura 4: Implementação da *feature* Som usando compilação condicional

3.2 Aspectos

A Figura 5 apresenta um aspecto que modulariza a *feature* de som utilizando AspectJ. Este aspecto atua no código base da classe *Chuckie*. Inicialmente, o aspecto proposto introduz um atributo do tipo *PlaySounds* nessa classe (linha 2) e declara conjuntos de pontos de junção que interceptam os métodos de pulo da classe (linhas 3-11). O aspecto também declara um conjunto de junção que intercepta instâncias de objetos da classe *Chuckie* (linhas

12-13). Um *advice* associado a esse conjunto de junção inicializa o atributo que foi introduzido na classe (linhas 14-16). O aspecto proposto contém também um segundo *advice*, que executa o som de pulo antes dos pontos de junção que denotam execuções de métodos de pulo da classe *Chuckie* (linhas 17-23).

Na implementação baseada em aspectos não foi necessário criar um método de execução de som, como no caso da implementação baseada em compilação condicional. O motivo é que o corpo desse método encontra-se contido no *advice* das linhas 17-23. Além disso, na solução apresentada foi possível modularizar integralmente a *feature* de Som em um único aspecto, evitando-se assim o seu entrelaçamento no código base do sistema.

```

1: public aspect Sound {
2:   PlaySounds Chuckie.playJump();
3:   pointcut soundJump():
4:     execution (* Chuckie.jump());
5:   pointcut soundJumpDir():
6:     execution (* Chuckie.jump_right());
7:   pointcut soundJumpLeft():
8:     execution (* Chuckie.jump_left());
9:   pointcut soundJumps(Chuckie o):
10:    (soundJump() || soundJumpDir() ||
11:    soundJumpLeft()) && target(o);
12:   pointcut soundJumpsCreate(Chuckie o):
13:     execution (Chuckie.new(..) && target(o);
14:   after(Chuckie o): soundJumpsCreate(o){
15:     o.playJump= new PlaySounds(...);
16:   }
17:   before(Chuckie o): soundJumps(o){
18:     try {
19:       o.playJump.play();
20:     } catch (Exception e) {
21:       ...
22:     }
23:   }
24: }

```

Figura 5: Implementação da feature Som usando aspectos

3.3 AHEAD

A Figura 6 mostra o refinamento da classe *Chuckie* responsável pela implementação da *feature* de Som. O refinamento mostrado é semelhante a uma classe de Java. Porém, utiliza-se a palavra reservada *refines* (linha 1), a qual identifica que o mesmo é um refinamento de uma classe já existente no sistema. O refinamento mostrado introduz na classe refinada um atributo para tratamento de som (linha 2). O construtor da classe base também é refinado para incluir a inicialização desse atributo (linhas 3-5). A exemplo da solução baseada em compilação condicional, também foi implementado um método, chamado *playJump*, responsável por executar o som de pulo (linhas 6-12). Por fim, para cada método que executa uma ação de pulo inclui-se uma chamada ao método *playJump* (linhas 14, 18 e 22). Nesses casos, para especificar que o código original do método base deve ser executado logo após o tratamento de som, utiliza-se a palavra reservada *Super* (linhas 15, 19 e 23).

4 Avaliação

Com base na experiência adquirida no estudo de caso descrito na Seção 3, apresenta-se a seguir uma avaliação das três tecnologias para implementação de variabilidades consideradas no trabalho. Essa avaliação é baseada nos seguintes critérios:

- **Configurabilidade.** Em desenvolvimento baseado em LPS, configurabilidade diz respeito à capacidade de se selecionar as *features* que serão incorporadas em um determinado produto (ou versão) da LPS. De acordo com esse critério, todas as três tecnologias permitem gerar de forma ágil versões do jogo *Chuckie Egg* com ou sem som. No caso de compilação condicional, basta ativar a diretiva de compilação utilizada para representar a *feature* Som. No caso de aspectos, basta combinar (ou não) o aspecto de som com o código base do sistema.

```

1: public refines class Chuckie {
2:   protected PlaySounds playJump;
3:   refines Chuckie(...){
4:     playJump= new PlaySounds(...);
5:   }
6:   protected void playJump() {
7:     try {
8:       playJump.play();
9:     } catch (Exception e) {
10:       ....
11:     }
12:   }
13:   public void jump() {
14:     playJump();
15:     Super.jump();
16:   }
17:   public void jump_right() {
18:     playJump();
19:     Super.jump_right();
20:   }
21:   public void jump_left() {
22:     playJump();
23:     Super.jump_left();
24:   }
25: }

```

Figura 6: Implementação da feature Som usando refinamentos

O mesmo ocorre com o refinamento de som apresentado na Figura 6, que pode ser combinado ou não com sua classe base.

- **Modularidade.** Na implementação baseada em CC, o código responsável pela implementação da *feature* Som fica entrelaçado no código base do sistema (conforme mostrado na Figura 4). Ou seja, não se obtém uma boa modularização e separação de interesses. Por outro lado, nas soluções baseadas em AOP e FOP, o código da *feature* Som é implementado em um módulo distinto. Com isso, não “polui-se” o código base com código relativo à implementação de interesses que não são mandatórios no sistema, como é o caso de efeitos sonoros.
- **Reusabilidade.** Essa característica diz respeito à capacidade de se reutilizar o código de uma *feature* em outros sistemas. Segundo o estudo de caso realizado, trata-se de um ponto negativo de todas as três tecnologias analisadas. Usando compilação condicional, o código da *feature* não é implementado em um módulo distinto, que possa ser reusado em outros sistemas. Por outro lado, apesar disso ocorrer nas soluções baseadas em AOP e FOP, os novos módulos criados referenciam elementos do código base do sistema. Em outras palavras, existe um acoplamento entre tais módulos e as classes do sistema *Chuckie Egg*. Por exemplo, no aspecto da Figura 5, existem referências a diversos membros da classe *Chuckie*, como os métodos *jump*, *jump_right* e *jump_left*. Referências semelhantes ocorrem no refinamento apresentado na Figura 6. A existência de tais acoplamento impede a reutilização dos aspectos e refinamentos mostrados em novos sistemas.
- **Simplicidade e facilidade de aprendizagem.** Dentre as três tecnologias consideradas, compilação condicional é a mais simples e de mais fácil domínio. Por outro lado, com base na experiência realizada, considera-se que FOP/AHEAD é uma tecnologia mais simples que AOP/AspectJ. Por exemplo, em AHEAD não é necessário declarar conjuntos de junção, definir o tipo de adendo (*after*, *before* ou *around*), associar *advices* a conjuntos de junção etc. Em vez disso, um refinamento tem acesso direto ao ambiente da classe refinada, o que simplifica a sua sintaxe.

- **Tamanho do código.** A Tabela 1 apresenta informações sobre o tamanho das três versões do jogo *Chuckie Egg*. Conforme pode ser observado nessa tabela, o maior acréscimo de linhas de código – em relação à versão original do jogo – ocorre na versão baseada em compilação condicional (+145 LOC), seguida pela versão baseada em FOP (+ 120 LOC) e depois

pela versão baseada em AOP (+98 LOC).

	LOC	Acréscimo
Versão original	1519	–
Versão baseada em CC	1664	+ 145
Versão baseada em AOP	1617	+ 98
Versão baseada em FOP	1639	+ 120

Tabela 1: Tamanho (em LOC)

A Tabela 2 resume os principais resultados da avaliação realizada. Conforme pode ser observado nessa tabela, apesar de o jogo Chuckie Egg ser um sistema bastante simples, pode-se concluir que existem benefícios inequívocos no uso de FOP e AOP para implementação de variabilidades típicas de jogos para celulares. Essas vantagens são mais claras quando se compara FOP e AOP com tecnologias mais tradicionais (e mais largamente utilizadas), como compilação condicional. Comparando-se especificamente FOP/AHEAD com AOP/AspectJ, considera-se que a principal vantagem do sistema AHEAD reside na simplicidade de sua linguagem para separação de interesses. Por outro lado, a principal desvantagem de AHEAD é a inexistência na linguagem de recursos de quantificação. Um refinamento sempre estende o comportamento de um método de uma classe do programa base. Já em AspectJ, recursos de quantificação permitem definir adendos que atuarão em múltiplos pontos de junção do programa base, o que é particularmente útil para implementação de requisitos transversais homogêneos (isto é, requisitos que requerem a implementação de um mesmo código em múltiplos pontos de um sistema).

	CC	AOP	FOP
Configurabilidade	+	+	+
Modularidade	-	+	+
Reusabilidade	-	-	-
Simplicidade	+	-	+/-
Tamanho do código	-	+	+/-

Tabela 2: Comparação entre as tecnologias CC, AOP e FOP

5 Trabalhos Relacionados

Em um trabalho anterior, documentamos uma experiência de extração de uma LPS na área de jogos para celulares [Santos e Valente 2008]. As principais diferenças para o presente trabalho são as seguintes: (1) a experiência foi realizada com um outro jogo (chamado Bomber³); (2) a experiência envolveu uma única tecnologia para implementação de variabilidades (programação orientada por *features*). Assim, no presente trabalho, procurou-se de forma integrada – isto é, utilizando o mesmo sistema alvo – mostrar os princípios de funcionamento, os benefícios e as principais limitações de duas outras tecnologias: programação orientada por aspectos e compilação condicional.

Batory cita o emprego de programação orientada por *features* na implementação de famílias de produtos envolvendo gerenciadores de bancos de dados, protocolos de redes, controladores de robôs e sistemas de controle de aeronaves [Batory 2004]. No entanto, tais experiências foram baseadas no sistema de FOP chamado GenVoca (do qual AHEAD é uma evolução). Alves et al. descrevem um conjunto de estratégias para migrar uma LPS implementada usando compilação condicional para aspectos, usando para isso um jogo para celular [Alves et al. 2006]. Mostram ainda que alguns padrões de variações não podem ser migrados para aspectos. Kastner, Apel e Batory descrevem o uso de AspectJ para refatorar um sistema gerenciador de bancos de dados (Oracle Berkeley DB) em uma linha de produtos [Kästner et al. 2007]. Assim como Alves, apontam também algumas limitações de aspectos quando usados com esse objetivo.

Com objetivo similar ao deste artigo, Lopez-Herrejon, Batory e Cook avaliam diversas tecnologias para implementação de *features*, incluindo não apenas AspectJ e AHEAD, mas também outros sistemas e linguagens de programação, como Hyper/J, Jiazzi

e Scala [Lopez-Herrejon et al. 2005]. No entanto, a avaliação realizada é baseada em um LPS hipotética, consistindo de uma calculadora com diferentes operações (adição, subtração etc) e formatos de números (números inteiros, decimais etc). Por outro lado, no presente artigo procurou-se demonstrar e comparar o uso de tecnologias para implementação de variabilidades utilizando um sistema simples, mas real, no domínio de jogos para celulares.

6 Considerações Finais

Neste artigo, realizou-se uma comparação entre três técnicas para modularização de variabilidades em jogos para celulares. Como estudo de caso, utilizou-se um jogo para celular implementado em J2ME (Chuckie Egg). Uma nova *feature* para tratamento de som foi acrescentada a esse jogo utilizando três diferentes tecnologias: compilação condicional, programação orientada por aspectos e programação orientada por *features*. Na comparação realizada, fica claro que tecnologias modernas de implementação de variabilidades – como aspectos e refinamentos – oferecem ganhos importantes em relação a tecnologias mais tradicionais, como compilação condicional. Esses ganhos ocorrem basicamente em critérios como modularidade e tamanho final do código. Como trabalho futuro, pretende-se estender a LPS gerada no artigo, investigando novas *features*. Pretende-se também utilizar essas técnicas para modularizar *features* típicas de jogos para consoles e PCs.

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG e CNPq.

Referências

- ALVES, V., NETO, A. C., SOARES, S., SANTOS, G., CALHEIROS, F., NEPOMUCENO, V., PIRES, D., LEAL, J., AND BORBA, P. 2006. From conditional compilation to aspects: A case study in software product lines migration. In *1st GPCE Workshop on Aspect-Oriented Product Line Engineering (AO-PLE)*, 1–6.
- BATORY, D., SARVELA, J. N., AND RAUSCHMAYER, A. 2003. Scaling step-wise refinement. In *25th International Conference on Software Engineering (ICSE)*, 187–197.
- BATORY, D. 2004. Feature-oriented programming and the AHEAD tool suite. In *26th International Conference on Software Engineering (ICSE)*, 702–703.
- CLEMENTS, P., AND NORTHROP, L. M. 2001. *Software Product Lines : Practices and Patterns*. Addison-Wesley.
- KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., LOPES, C., LOINGTIER, J.-M., AND IRWIN, J. 1997. Aspect-oriented programming. In *11th European Conference on Object-Oriented Programming (ECOOP)*, Springer Verlag, vol. 1241 of *LNCS*, 220–242.
- KICZALES, G., HILSDALE, E., HUGUNIN, J., KERSTEN, M., PALM, J., AND GRISWOLD, W. G. 2001. An overview of AspectJ. In *15th European Conference on Object-Oriented Programming (ECOOP)*, Springer Verlag, vol. 2072 of *LNCS*, 327–355.
- KÄSTNER, C., APEL, S., AND BATORY, D. 2007. A case study implementing features using AspectJ. In *11th International Software Product Line Conference (SPLC)*, 223–232.
- LOPEZ-HERREJON, R., BATORY, D., AND COOK, W. R. 2005. Evaluating support for features in advanced modularization technologies. In *19th European Conference on Object-Oriented Programming (ECOOP)*, Springer-Verlag, vol. 3586 of *LNCS*, 169–194.
- SANTOS, R., AND VALENTE, M. T. 2008. Extração de uma linha de produtos de software na área de jogos para celulares usando programação orientada por *features*. In *II Latin American Workshop on Aspect-Oriented Software Development (LA-WASP)*, 1–10.

³Disponível em <http://j2mebomber.sourceforge.net>.