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
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 plataform 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 provêem 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.

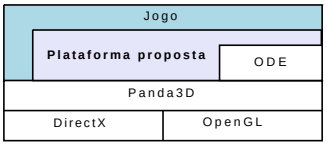

Figura 1: *Plataforma proposta.*

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.

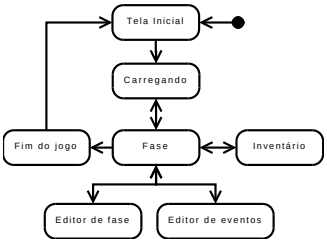

Figura 2: *Máquina de estados geral do jogo.*

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.

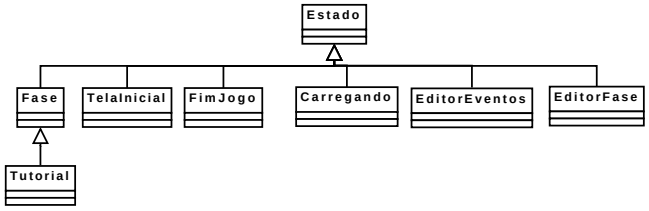

Figura 3: *Classes que representam os estados do jogo.*

A plataforma já implementa várias funcionalidades básicas de um jogo, como pode ser visto na Figura 4.

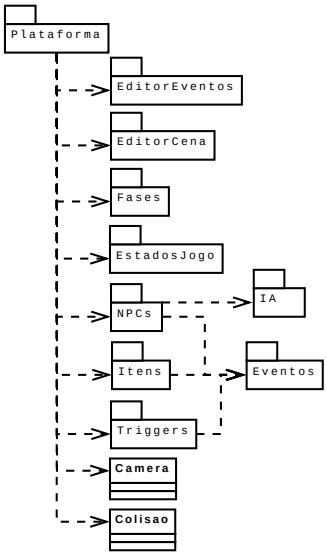

Figura 4: *Pacotes.*

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.

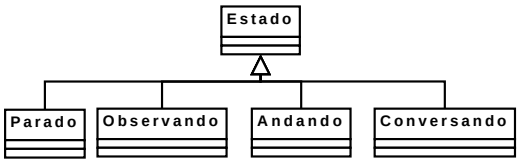

Figura 5: *Classes que representam os estados de uma pessoa.*

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 (*waypoints*), 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].

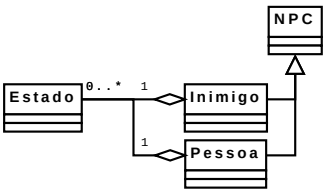

Figura 6: *Estrutura de classes para um NPC.*

2.1.2 Objetos do jogo

Os objetos visíveis na tela são do tipo das classes descritas na Figura 7.

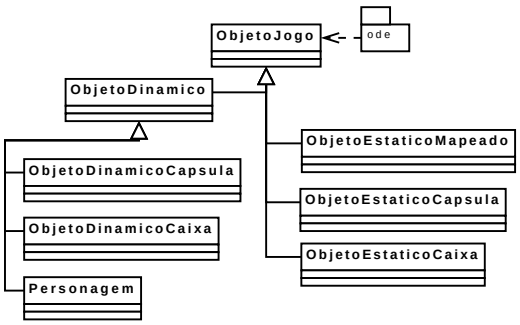

Figura 7: *Classes que representam os objetos inseridos no jogo.*

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 sucetível 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.

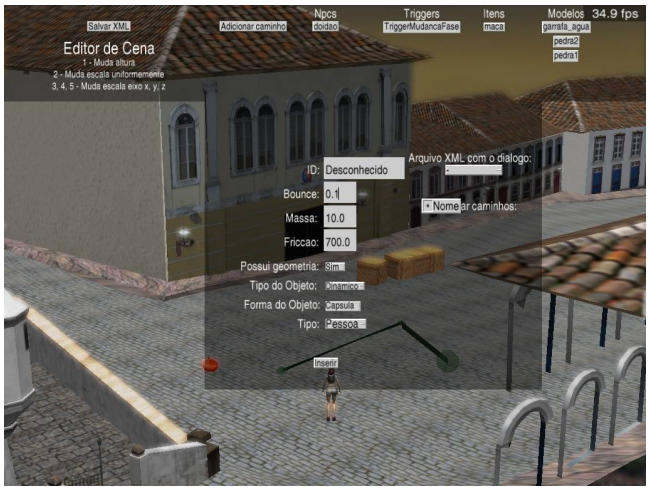

Figura 8: *Tela do editor de fases.*

Na Tabela 1 é exibido um arquivo XML como exemplo, onde há um personagem inserido na cena.

```
<cena>
  <npc>
    <id>vendedor</id>
    <dialogos/>
    <tipo>Pessoa</tipo>
    <estado>Parado</estado>
    <modelo>
      <arquivo>vendedor</arquivo>
      <dirBase>personagens/humanos/</dirBase>
      <posicao>Point3(246.322, -160.502, 157.358)</
          posicao>
      <escala>VBase3(0.0471012, 0.0471012, 0.0471012)</
          escala>
      <rotacao>VBase3(0, 0, 0)</rotacao>
      <visivel>Sim</visivel>
    </modelo>
    <geometria>
      <geoTipo>Dinamico</geoTipo>
      <geoForma>Capsula</geoForma>
      <friccao>700.0</friccao>
      <massa>10.0</massa>
      <bounce>0.1</bounce>
    </geometria>
    <caminho>
      <loop>Sim</loop>
      <ponto>Point3(246.632, -159.704, 127.387)</ponto>
      <ponto>Point3(272.94, -319.947, 117.257)</ponto>
      <ponto>Point3(203.891, -318.641, 116.292)</ponto>
    </caminho>
  </npc>
</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.

```
<eventos>
  <evento>
    <tipo>Plataforma.Eventos.Conversa.Conversa</tipo>
    <id>ConversaGuardinha</id>
    <preRequisitos>
      <npc>
        <idRef>guardinha</idRef>
      </npc>
    </preRequisitos>
    <xml>guardinha2.xml</xml>
  </evento>
  <evento>
    <tipo>Plataforma.Eventos.Conversa.Conversa</tipo>
    <id>ConversaVendedor</id>
    <preRequisitos>
      <npc>
        <idRef>vendedor</idRef>
      </npc>
    </preRequisitos>
    <xml>vendedor.xml</xml>
  </evento>
  <evento>
    <tipo>Plataforma.Eventos.AlteraFase.AlteraFase</tipo>
    <id>Galpao</id>
    <preRequisitos>
      <evento>
        <idRef>ConversaVendedor</idRef>
      </evento>
      <evento>
        <idRef>ConversaGuardinha</idRef>
      </evento>
      <trigger>
        <idRef>Fases.Galpao.Galpao</idRef>
      </trigger>
    </preRequisitos>
  </evento>
</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.

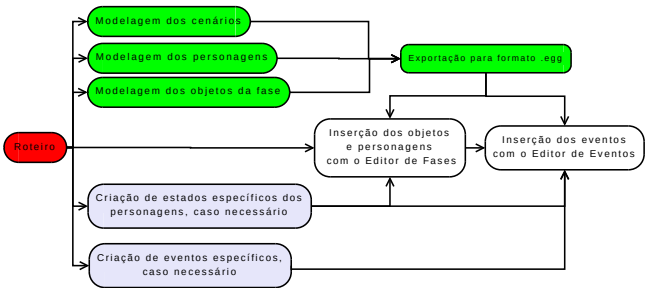

Figura 10: *Fluxo de desenvolvimento.*

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 VLISSIDES, 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.*