Fuzzy River Raid - Uma Abordagem Adaptativa para Navegação Autônoma

André B. Boscatto* Gilberto Nakamiti Carlos Tobar Toledo

Pontifícia Universidade Católica de Campinas, Faculdade de Engenharia de Computação, Brasil

Figura 1: Exemplo de tela do jogo *Fuzzy River Raid*

Resumo

Este trabalho apresenta um sistema de controle tempo-real para uma nave autônoma, cuja estratégia de controle inclui teoria dos conjuntos nebulosos (*fuzzy*), algoritmos genéticos e sistemas baseados em casos. A estratégia de controle é testada em um cenário similar ao de um jogo clássico, denominado *River Raid*, onde uma nave deve voar sobre um desfiladeiro sem chocar-se com obstáculos.

Palavras-chave: controle *fuzzy*, sistemas híbridos, algoritmos genéticos, raciocínio baseado em casos, River Raid.

Abstract

This paper presents a real-time control system for an autonomous airplane. Its control strategy includes fuzzy sets, genetic algorithms, and case-based systems. Tests in a *River-Raid*-like environment, where the airplane flies over a river avoiding obstacles, are shown.

Keywords: fuzzy control, hybrid systems, genetic algorithms, case-based systems, River Raid.

Contato:
*andreboscatto@gmail.com
{nakamiti,tobar}@puc-campinas.edu.br

1. Introdução

A movimentação automática de objetos é essencial em vários tipos de jogos, tanto para objetos aliados do usuário quanto para objetos inimigos. Os usuários cada vez mais esperam por surpresas, que, em fases mais avançadas do jogo, podem tornar os desafios ainda mais instigantes. Caso os objetos movimentem-se da mesma forma em cada cena, o jogo pode tornar-se repetitivo e previsível. Além disso, é desejável que objetos aliados possam aprender e adaptar-se aos movimentos do usuário.

Este trabalho apresenta um sistema inteligente de navegação automática, que utiliza cenários baseados no jogo *River Raid* para apresentar mecanismos adaptativos e de aprendizagem. Para tanto, são utilizados a teoria dos conjuntos nebulosos (*fuzzy*), algoritmos genéticos (AG) e raciocínio baseado em casos (RBC). Um exemplo de tela do sistema é mostrado na Figura 1.

A teoria dos conjuntos nebulosos é conhecida por propiciar que os sistemas manipulem de forma eficaz informações incertas ou incompletas, como é o caso da movimentação dos demais objetos em uma cena. Os sistemas baseados em casos permitem o reuso do esforço computacional, reaproveitando decisões anteriores em situações similares, característica desejável em sistemas reativos e/ou tempo-real. Como os sistemas baseados em casos reusam casos anteriores adaptando-os às novas situações, seu desempenho pode levar a ótimos locais, quando soluções melhores podem não ser alcançadas se suas características não estiverem presentes em nenhum caso armazenado. Para lidar com essa restrição, são usados algoritmos genéticos, cujos mecanismos (em particular a mutação) permitem gerar soluções não armazenadas anteriormente.

O sistema ainda apresenta uma camada de simulação, usada para predizer as consequências de uma determinada ação antes que ela seja tomada. Possíveis ações são simuladas por *threads* executadas em *background* e seus resultados são usados como função de *fitness* do algoritmo genético.

2. Trabalhos Correlatos

Há vários sistemas e abordagens para a simulação de veículos, como por exemplo os apresentados por Osório [2009], e vários deles utilizam técnicas de Inteligência Artificial (IA).

Garcia et al. [2009] apresenta um sistema de planejamento de rotas utilizando teoria dos conjuntos nebulosos para guiar robôs (formigas), onde o método de decisão leva em consideração especialmente a distância entre a origem e o destino. Khatoon et al. [2012] apresenta um sistema *fuzzy* para evitar colisões com robôs autônomos.

Algoritmos genéticos também já foram usados em sistemas de navegação, como por exemplo em Manikas [2007].

XaiTTRAFFIC [2010] é um simulador de tráfego com algoritmos de colisão e interface gráfica 2D, embora não conte com funcionalidades como movimentação de obstáculos em grupos. *XaiTMOVE* [2010] é um simulador para naves autônomas, que utiliza técnicas de IA para gerar caminhos, permitindo a inclusão de obstáculos diversos, em um ambiente 2D.

Nakamiti et al. [2011] apresenta um sistema baseado em agentes que utiliza as mesmas técnicas de IA (teoria dos conjuntos nebulosos, algoritmos genéticos e raciocínio baseado em casos). As técnicas são usadas por um controlador tempo-real, que verifica as posições de outros objetos para gerar decisões de controle em um controlador de tráfego urbano.

3. Arquitetura do Sistema

A arquitetura do sistema é mostrada na Figura 2, consistindo de diversos módulos. O módulo XNA utiliza as bibliotecas da plataforma de mesmo nome, baseada no *framework* .NET, em particular, os métodos *Load()*, *Unload()*, *Update()* e *Draw()*. Esses métodos são utilizados para carregar os dados para a memória principal (e assim conseguir melhor desempenho), limpar a memória, atualizar as posições dos objetos e desenhar a tela (cenário).

O módulo *Preparar Obstáculo* consiste em obter as informações do cenário (os obstáculos e suas respectivas coordenadas).

O módulo *Atualizar Posição da Nave* é responsável por obter a movimentação da nave e é dividido em três métodos: um para movimentar a nave, que atualiza a posição da nave principal a partir dos comandos de alteração na direção e intensidade de movimento; um método para checar os limites da tela e impedir a movimentação for a dos limites estabelecidos; e um que checa a colisão com os objetos do cenário e intensidade observando a sobreposição dos pixels para maior precisão em relação aos objetos não retangulares (algoritmos disponível em Perpixel [2010]).

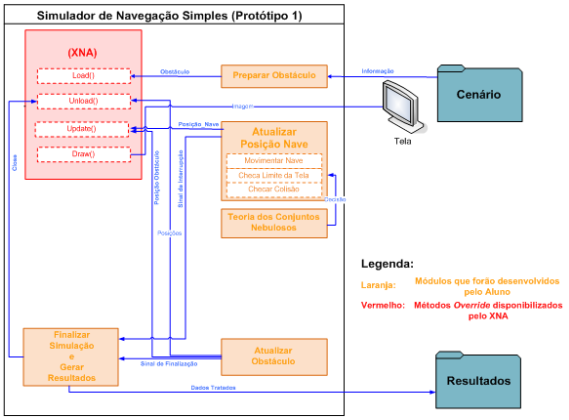

Figura 2: Arquitetura do sistema

O módulo *Atualizar Obstáculo* tem a função de deslocar obstáculos da tela para baixo, dando a impressão de que o cenário está em constante movimento, fazendo com que a nave pareça se deslocar para a frente.

O módulo *Finalizar Simulação e Gerar Resultados* libera da memória todos os objetos que foram utilizados durante a simulação e gera um arquivo texto com sua análise, contendo informações como o número de objetos ultrapassados e o tempo total da simulação.

Finalmente, os módulos *Teoria dos Conjuntos Nebulosos, Raciocínio Baseado em Casos* e *Algoritmos Genéticos* são os responsáveis pelo comportamento adaptativo da nave e serão detalhados a seguir.

3.1 MÓDULO FUZZY

O módulo *Teoria dos Conjuntos Nebulosos* (*Fuzzy*) realiza o treinamento do sistema, gerando a base inicial de casos que será utilizada pelos outros módulos adaptativos.

Basicamente, ele converte em variáveis linguísticas as distâncias entre a nave e os demais objetos da cena, usando a função de pertinência *Distância* mostrada na Figura 3.

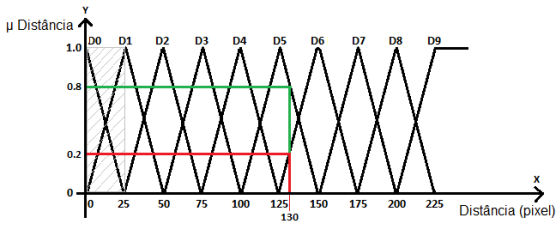

Figura 3: Representação Gráfica da Variável Distância.

As funções D0 a D9 são utilizadas no lugar de termos linguísticos como *Baixa, Média, Alta*, etc. Na Figura 3, um objeto que dista 130 pixels da nave (distância cartesiana) teria como função de pertinência 0,8 D5 e 0,2 D6. A aplicação das regras *fuzzy* e o cálculo da ação a ser tomada é obtido através do centro de área, descrito na literatura, incluindo em Nakamiti et. al. [2001]. A fase de treinamento *fuzzy* é executada até que gere 50 casos (valor configurável), que servirão de entrada ao mecanismo híbrido genético-baseado em casos, facilitando sua convergência.

3.2 MÓDULO RACIOCÍNIO BASEADO EM CASOS

Para o armazenamento e recuperação de casos, foi proposta uma estrutura de dados para armazenar cada caso, ilustrada pela Figura 4. Nessa estrutura são armazenadas informações como número de objetos na cena, distância e posição relativa (à nave). Como a estrutura é fixa, quando houver mais de quatro objetos à direita (ou à esquerda) da nave, somente as posições dos quatro mais próximos são considerados. A decisão gerada (direção adotada pela nave), bem como eventuais colisões depois de 1, 3, 5 e 10 segundos também são armazenadas.

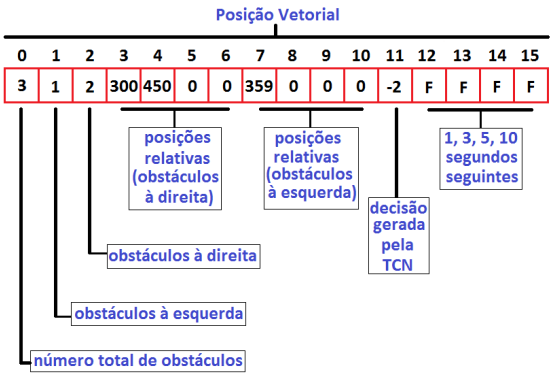

Figura 4: Estrutura de um caso

A previsão sobre eventuais colisões é de fundamental importância na tomada de decisões. Neste tipo de aplicação, a nave deve alterar sua direção de forma a tentar evitar colidir com outros objetos da cena. Para isso, foi implementado um simulador simples, que toma a cena inicial e simula sua continuação de forma acelerada, através de uma *thread* executada em *background*. Através dessa estratégia pode ser possível antecipar, com relativa precisão, os resultados de ações, considerando que os outros objetos continuarão em suas trajetórias. Mais detalhadamente, cada cena inicial é executada como em uma estrutura de pipeline, que armazena o resultado da simulação na estrutura do caso (colidiu ou não colidiu) após 1,3,5 e 10 segundos (Figura 5).

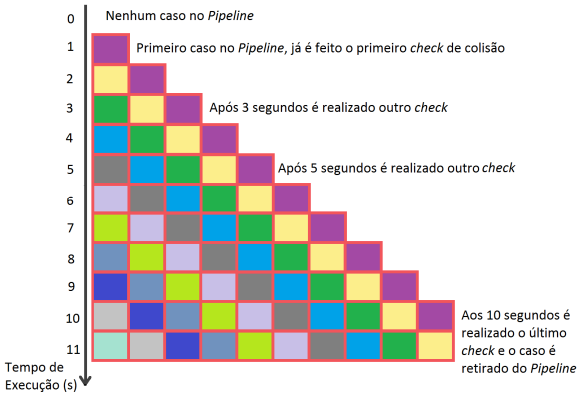

Figura 5: Simulação em *pipeline*.

3.3 MÓDULO ALGORITMOS GENÉTICOS

Para a tomada de decisão a partir de um dado cenário (direção a ser tomada pela nave), o primeiro passo é a recuperação dos casos mais similares, a partir da base de casos. Essa busca é realizada por uma função simples de similaridade baseada em distância. Os casos recuperados (todos até um *threshold*) constituem a primeira geração a ser usada pelo algoritmo genético. Os casos (indivíduos) são escolhidos a partir de sua medida de *fitness*, que indica se são mais ou menos adequados à situação.

Para o cálculo da aptidão (*fitness*) de cada caso, foi usada a simulação descrita anteriormente, onde cada *thread* era executada por um segundo, gerando e testando cerca de 120 casos, executados em um notebook padrão (A execução da tomada de decisões *fuzzy*, na fase de treinamento, era executada muito mais rápido, por tratar-se da verificação e aplicação de regras de produção.) Os resultados de diferentes decisões, que podem causar ou não colisões, são ilustrados através de uma montagem de cena, na Figura 6.

Figura 6: Ilustração da simulação de possíveis trajetos

Mecanismos usuais de *crossing* over e mutação são aplicados. Mais detalhes podem também ser obtidos em Nakamiti [2011].

4. Resultados

Para testar a adaptabilidade de sistema, foram propostos cinco cenários, com graus crescentes de complexidade, compostos por quatro a dezenove obstáculos. Nos primeiros cenários, os obstáculos foram dispostos com espaçamento suficiente para que a nave pudesse passar entre eles, sem colisão. Nos últimos cenários, a nave devia desviar-se constantemente, com os obstáculos dispostos em linha, coluna e nas diagonais, uns em relação aos outros.

O sistema foi testado usando somente as regras de produção *fuzzy* e depois também completo, com o mecanismo baseado em casos e algoritmos genéticos, para verificar a aprendizagem. Cada teste foi realizado dez vezes, tendo a posição inicial da nave sido escolhida aleatoriamente na linha inferior do cenário.

Em todos os testes em que utilizamos apenas as regras de produção *fuzzy*, o mesmo resultado foi obtido (a nave conseguiu ou não atravessar o cenário sem colisões). Cabe observar que usamos regras de produção estáticas, isto é, não geramos novas regras em tempo de execução. Nos testes com o sistema completo, o sistema colidiu com os obstáculos nos cenários mais complexos nas primeiras tentativas. Depois, em todos os casos, o sistema aprendeu uma solução e prosseguiu nos testes sem colidir.

A Tabela 1 mostra os resultados obtidos, com o número de tentativas sem colisão para cada cenário.

Cenário (obstáculos)	Fuzzy	Completo
1. Quatro obstáculos dispostos próximos aos cantos da tela (cenário).	10	10
2. Nove obstáculos divididos em três linhas, alternadamente do lado esquerdo e direito do cenário.	10	10
3. Nove obstáculos dispostos em *zig-zag*	10	10
4. Dezoito obstáculos, que incluem os obstáculos dos cenários 2 e 3.	0	7
5. Dezenove obstáculos, sendo catorze vindos do cenário anterior, quatro obstáculos formando pirâmides com os anteriores e mais um obstáculo disposto aleatoriamente.	0	4

5. Considerações Finais

Este trabalho apresentou uma estratégia adaptativa para navegação autônoma, que inclui a teoria dos conjuntos nebulosos, os sistemas baseados em casos e os algoritmos genéticos.

A estratégia foi aplicada em um ambiente do tipo "*River Raid*", onde uma nave deve desviar de obstáculos, com níveis crescentes de dificuldade.

Os resultados obtidos indicam que o sistema é capaz de adaptar-se a diferentes cenários e aprender com suas experiências anteriores, mesmo em cenários com muitos obstáculos e pouquíssimas alternativas de sucesso.

Referências

GARCIA, M. et al. 2009. Path planning for autonomous robot navigation with ant colony optimization and fuzzy cost funtion evaluation. *Applied Soft Computing* 9(3), 1102-1110.

KHATOON, S. 2012. Autonomous Mobile Robot Navigation by Combining Local and Global Techniques. *Intrenational Journal of Computer Applications* 37(3), 1-10.

MANIKAS, T. el al. 2007. Genetic Algoritms for Autonomous Robot Navigation. *IEEE Instrumentation and Measurement Magazine* 10(6), 26-31.

NAKAMITI, G., Silva V.E., Ventura, J.H., 2011. "Urban Traffic Control and Monitoring: An Approach for the Brazilian Intelligent Cities Project", *In: Anais do XI International Conference on Intelligent Systems and Knowledge Engineering", Dezembro de 2011 Xangai.* Berlin: Springer, 543-551.

OSORIO, F.S. et al. 2009. " Simulação Virtual de Carros em Jogos e Aplicações de IA". *In: Anais do VIII Brazilian Symposium on Games and Digital Entertainment". Outubro de 2009 Rio de Janeiro.* 257-306.

PERPIXEL. *Collision Series 2*: 2D Per-Pixel Collision. Disponível em: <http://create.msdn.com/downloads/?id=97&filename=PerPixelCollisionSample_4_0.zip> [Acessado em 08 novembro 2010].

XAITMOVE. Disponível em: http://www.xaitment.com/english/products/xaitmove2. html [Acessado em 05 junho 2010].

XAITTRAFFIC. Disponível em: http://www.xaitment.com/english/products/ [Acessado em 05 junho 2010].