Um método para geração de desvio de obstáculos em um jogo de corrida

Thiago Ribeiro de Azeredo Pedro Sampaio Vieira Ítalo de Oliveira Matias

Universidade Candido Mendes, Campos dos Goytacazes, Brasil

Resumo

Este artigo descreve uma implementação de desvio dinâmico de obstáculos para os personagens dotados de inteligência artificial aplicável a diversos tipos de jogos. Tal aplicação torna a experiência do jogo mais realista e desafiadora. O experimento levou à conclusão que regras simples geram resultados interessantes quanto a tomada de decisão dos personagens.

Keywords: inteligência artificial, desvio de obstáculos

Authors' contact:
{thiagoribeiro,pesvieira}@gmail.com
italo@ucam-campos.br

1. Introdução

A busca pela melhor experiência em jogos eletrônicos é constante. Um jogo eletrônico é um dos raros aplicativos onde é possível utilizar uma vasta diversidade de áreas da computação em conjunto, a fim de se criar um ambiente realista onde é necessário não somente de uma boa qualidade gráfica, mas também de comportamentos realistas dos personagens não controlados pelos jogadores, NPC (*non player character*), tanto relacionados à física quanto às suas ações.

A evolução do realismo gráfico dos jogos atuais está bastante avançada e influencia diretamente a aplicação das técnicas de Inteligência Artificial em um jogo. A cada novo jogo lançado, percebe-se um grande aumento no nível de detalhamento gráfico, com isso, surge uma grande cobrança para que a inteligência do jogo consiga acompanhar essa evolução, já que o realismo no comportamento dos personagens NPC contribui fortemente para que o jogador se sinta ainda mais imerso no ambiente virtual.

Sendo assim, este trabalho descreve uma proposta de implementação de desvio dinâmico de obstáculos aplicável em diversos estilos de jogos. Foi escolhido o jogo de corrida pelo aspecto competitivo e por utilizar toda a base já criada durante o trabalho realizado em [Azeredo 2006].

Para tal, a organização desse artigo é feita da seguinte maneira: a seção 2 mostra as ferramentas que foram utilizadas nesse trabalho; a próxima seção traz uma breve abordagem sobre as técnicas para o controle do carro; em seguida a técnica de desvio de obstáculos é apresentada; Na seção 5 são feitas as conclusões finais, além do registro de trabalhos futuros.

2. Ferramentas

Para auxiliar o desenvolvimento, algumas ferramentas foram utilizadas durante o trabalho de [Azeredo et al., 2006] a fim de automatizar tarefas. Sendo assim, tais ferramentas são apresentadas nessa seção.

É importante destacar que todas as ferramentas escolhidas são multiplataforma, o que permitiu que todo o projeto fosse migrado para ambiente Linux, apesar de sua versão inicial ter sido criada sobre o ambiente Windows.

2.1 Engine Física

Uma engine física é responsável por todos os cálculos físicos presentes no jogo, como: massa, força gravitacional, atrito, elasticidade, juntas (articulações), colisão entre objetos, etc. Tais engines são amplamente usadas na indústria de jogos, cinema e na criação de simuladores, com o objetivo de proporcionar mais realismo.

A engine física neste trabalho é responsável por todo o cálculo que envolve o comportamento dos veículos na pista de corrida.

A engine física ODE foi escolhida neste trabalho por ser uma biblioteca gratuita (licença GNU LGPL / BSD) e de alta qualidade, com grande utilização em aplicações de realidade virtual. É rápida, flexível, robusta e independente de plataforma.

2.2 Engine Gráfica

Uma engine gráfica é a responsável pelos cálculos que envolvem toda a parte visual da aplicação. Trabala direto na renderização dos objetos de cena, criando luzes, sombras, rotacionando e transladando objetos, bem como outras técnicas de computação gráfica. Para isso utilizam-se artifícios para otimizar a quantidade de quadros gerados por segundo, sendo não apenas

utilizadas na construção de jogos, como também aplicações em ambientes de simulação computacional.

Este trabalho optou-se pela utilização da engine gráfica Ogre3D. Trata-se de uma open scene-oriented em C++ (biblioteca orientada a cena), onde todos os objetos que devem ser exibidos na tela estão ligados a uma cena. Sua escolha deveu-se por sua estrutura ser bastante flexível, projetada para ser fácil e intuitiva para os desenvolvedores, utilizando tanto o DirectX como o OpenGL como API gráfica. Alem de ser multiplataforma e gratuita (licença GNU LGPL). Vem sendo desenvolvida por uma comunidade bastante ativa, possuindo resultados bastante satisfatórios e uma documentação bem ampla e objetiva.

Um outro fator que auxiliou na escolha do Ogre3D é a sua gama de plugins, onde se pode encontrar o OgreODE, capaz de integrar o ODE ao sistema gráfico criando um mundo virtual dinâmico, onde em poucas linhas de código é possível ter objetos interagindo na cena [OGRE 2000b]. Além deste benefício, o OgreODE traz implementado um modelo de veículos com roda, podendo ser criados desde monociclos até caminhões e tratores. Os parâmetros do veículo são: suspensão, amortecimento, massa, centro de gravidade, torque, atrito, força de frenagem, localização das rodas, entre outros.

3. Estratégias de Controle do Carro

3.1. O Problema

O problema consiste em um simulador de corrida no qual há a necessidade de controlar todos os movimentos de um veículo NPC durante a prova. O realismo de tais ações é muito importante para proporcionar uma maior competitividade na corrida, fazendo com que o jogo se torne mais atrativo ao jogador.

3.2. Direção do Carro

Controlar o carro durante uma corrida é um grande desafio, tanto em uma situação real quanto numa simulação computacional. O controle dos NPCs (neste caso os carros controlados pela IA) não é diferente. Apesar de haver técnicas que visam apenas passar a sensação de que os oponentes estão correndo (como, por exemplo, reposicionar o carro na cena em pontos pré-determinados), há outras que buscam simular os pilotos, como se fossem pilotos automáticos.

Para o controle direcional do carro neste trabalho, partiu-se dessa idéia de criar um piloto automático. Desta forma o carro enfrenta todos os desafios enfrentados por um jogador humano como, por exemplo, completar uma volta no menor tempo possível, desviar de obstáculos, saber o momento de acelerar e frear, entre outros. Sendo assim o carro (NPC) deve primeiramente "conhecer" a pista.

3.2.1. Representação do Trajeto em um Ambiente Bidimensional

Esta etapa resume-se em representar a pista como uma estrutura de simples visualização para o piloto, fazendo com que o mesmo se localize na pista. Isto é importante na manutenção do carro sempre dentro da pista, pois todas as decisões são tomadas a partir da análise de dados baseados nessa estrutura.

Assim, há a necessidade de uma representação da pista que seja consistente e que não prejudique o processamento. Para isto, foram utilizados [BIASILLO 2002a; BIASILLO 2002b] que sugerem a representação da pista por setores, em que cada setor é um paralelogramo.

Os pontos que representam os vértices dos setores no espaço são fixados sobre a pista, como mostrado na figura 1. Os setores da pista sempre compartilham dois lados com outros setores, este compartilhamento foi denominado interface de conexão. Estas interfaces interligam o setor *X* com o setor *X-1* e *X+1*.

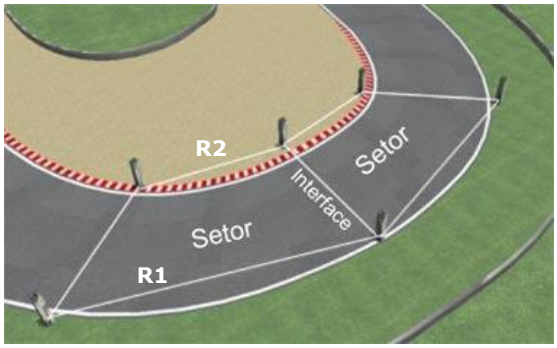

Figura 1: Demonstração dos paralelogramos dos Setores.

Cada interface é um segmento de reta, e desta forma, ela é representada por dois pontos (A e B) que definem a sua localização. Para facilitar os cálculos e reduzir o custo do processamento, a posição no eixo Y (que representa a altura) é desconsiderada, assim os pontos serão formados pelo eixo X e Z.

O critério para o posicionamento dos setores na pista é definido como:

- Tão distante da interface anterior quanto possível, de forma que se forem traçadas retas (*R1* e *R2*) entre os pontos extremos da interface anterior e atual, estas retas estejam dentro da pista conforme figura 1.
- Manter as interfaces sempre perpendiculares à lateral da pista. E nas curvas, mantê-las perpendicular à tangente.

3.2.2. Guiando para um ponto

Fazer o piloto automático guiar o carro para um ponto *P* requer inicialmente ter tal ponto. Tendo uma pista inteira, há a necessidade de ter vários destes pontos. Dessa maneira, a estrutura dos setores foi utilizada para armazená-los. Foi definido que os pontos guia (pontos definem o traçado imaginário), deveriam pertencer ao segmento de reta de uma interface. Desta forma, em cada setor o carro deve guiar para o ponto guia da interface diretamente a sua frente.

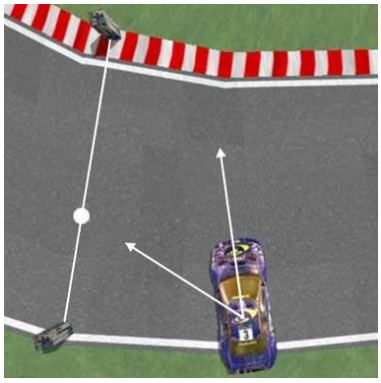

Figura 2: Vetores para o cálculo do ângulo de virada dos pneus.

Para o carro ser guiado para um ponto, o piloto automático precisa fazer com que o carro vire as rodas em direção ao seu destino. Para isso, foi utilizado o cálculo vetorial, para determinar qual o ângulo que as rodas devem girar. Conforme o algoritmo 1. Este procedimento é executado a cada passo para corrigir a direção.

Algoritmo 1: Calcular ângulo [ADZIMA 2002].

```
01 direção = alvo – posição_atual;
02 fX = ProdutoInterno(direção, X_ori);
03 fZ = ProdutoInterno(direção, Z_ori);
04 retornar ArcoTangente(fX, fZ);
```

onde *alvo* representa a posição do ponto guia, *posição atual* é a posição do carro, *X_ori* é o vetor de orientação do carro no eixo X e *Z_ori* é o vetor de orientação do carro no eixo Z.

3.2.3. Suavização do traçado

Nesta parte do desenvolvimento, ao carregar os pontos guia de um arquivo, o carro já anda sozinho na pista, porém de uma forma bastante precária. Um dos problemas neste momento é a mudança brusca do ponto guia do carro ao se trocar de setor, fazendo com que as rodas virem rapidamente, levando o carro a perder o controle.

Este problema é minimizado adicionando uma função que interpola o ponto guia atual e o próximo. Assim, o piloto automático deixa de guiar para um ponto (forma discreta) em cada setor para guiar para um ponto diferente a cada momento (forma contínua). Esta função é definida como:

$$(angulo1*dist) + p*(angulo2*(1-dist)),$$

onde *angulo1* é o quanto a roda deve virar para passar no ponto guia da interface a frente (como apresentado anteriormente), *angulo2* é o quanto a roda deve virar para passar no próximo ponto guia, *dist* é o percentual (de 0 a 1) da distância percorrida pelo carro dentro do setor atual, e *p* é um peso (entre 0 e 1) dado para diminuir a influência do próximo ponto guia.

Com esta função, quanto mais próximo do ponto guia da interface à frente, menor sua influência e maior a influência do ponto guia seguinte. Evitando uma mudança súbita, levando o carro a manter-se estável e passando uma impressão mais humana na forma de condução do piloto automático.

4. Desvio de obstáculos

Para que seja possível efetuar o desvio de obstáculos, primeiramente é necessário detectá-los, e em seguida determinar a melhor forma de desviar.

Este capítulo apresenta a solução gerada para obter uma boa qualidade na tomada de decisão dos jogadores NPC em ocasiões de colisão eminente.

4.1. Detecção do obstáculo

Há duas formas básicas para detecção de obstáculos em sistemas autônomos. A primeira seria a onisciência, onde o piloto automático sabe previamente a posição exata dos obstáculos (adversários ou objetos) na pista. A outra forma seria a simulação do olho, onde o piloto só verá o obstáculo caso esteja dentro do seu ângulo de visão.

Tendendo sempre a resolver os problemas com base no dia-a-dia, este trabalho utilizou uma técnica para simular a função do olho humano. Para tal, um radar foi configurado com um ângulo de visão que variaram entre 90º e 120º durante os testes. Raios foram enviados de 3º em 3º para checar a região de interesse. Estes raios invisíveis são criados pela engine gráfica utilizando o método RaySceneQuery e têm como objetivo retornar os objetos da cena interceptados até uma determinada distância. Estes objetos são filtrados, já que objetos como placas e muros de proteção não devem ser levados em consideração, e então colocados numa estrutura seqüencial, como uma lista, e ordenados pela proximidade para posterior análise pelo sistema de desvio.

Como um jogo é uma aplicação que requer cuidados no custo computacional, ao invés de fazer a conferência completa de todo ângulo de visão a cada passo, foi convencionado que cada iteração é realizada em um passo e o tempo entre as iterações é controlado. Esta característica também gera uma margem de erro no sistema, já que o piloto pode demorar a ver o obstáculo. Porém isto é bem vindo, visto que

proporciona ao piloto automático a chance de cometer erros, como qualquer outro jogador.

Após a coleta dos objetos, é feito uma filtragem pela distância existente entre os dois corpos. Isto simula a distância mínima de reação, filtrando objetos muito distantes.

4.2. Desvio do obstáculo

O sistema de desvio baseia-se na análise dos dados diretamente expostos na lista de obstáculos, interpretando-os e tomando as decisões necessárias para evitar uma colisão.

Inicialmente é feita uma conferência na lista gerada pelo sistema de detecção de obstáculos em busca do objeto mais próximo que realmente estejam em rota de colisão. Isto é verificado através da análise feita entre uma reta na mesma direção do vetor de velocidade do carro (que indica em qual direção o carro está indo) e o *bounding-box* que envolve o objeto. Caso a reta corte o *bounding-box*, o sistema é acionado.

Confirmando que há um obstáculo em rota de colisão, sabendo o vetor de direção do veículo e a localização do obstáculo na cena, é feito uma análise para determinar o sentido da manobra levando em consideração dois fatores: a distância entre o veículo e o obstáculo, o espaço restante em cada lado da pista. Desta forma será escolhido o lado com maior espaço.

Após a seleção do lado que será feito o desvio, o ângulo de giro do volante é calculado baseado na distância entre o veículo e o obstáculo, onde a intensidade da manobra é diretamente proporcional à proximidade do obstáculo, de forma a evitar a colisão e esta informação entra como um fator de "distorção" no ângulo de movimentação sobre a linha-guia.

Desta forma, mesmo que o obstáculo esteja presente em uma curva, o veículo consegue desviar e prosseguir, pois o ângulo de direção não é arbitrariamente substituído, e sim corrigido.

4.3. Avaliação dos resultados

Como esperado, o sistema gerou comportamento variado durante os testes, sendo capaz de desviar de outros veículos para fazer ultrapassagens e se deslocando para o lado ao sofrê-las. Passando para o jogador a sensação de oponente racional.

Em poucos momentos se mostrou falho, gerando colisões e trazendo a variação nos movimentos tornando o sistema não-determinístico.

5. Conclusão

A adição da regra de desvio, mesmo com sua implementação simples, resultou em uma demonstração mais realista durante os testes, permitindo que o jogador obtivesse uma experiência mais imersiva e competitiva durante a corrida.

Foi observado um comportamento reativo no veículo controlado pelo NPC, que através do seu sistema de desvio de obstáculos, começou a promover ultrapassagens. Apesar desse comportamento não ter sido diretamente o objeto desta pesquisa, o fato dele ter "aparecido" de forma satisfatória é um fator importante que deve ser destacado. O que permite um trabalho futuro nessa direção para aprimorar essa ação do NPC, que aumentará ainda mais o nível de realismo da aplicação.

Referências

ADZIMA, J. Competitive AI Racing under Open Street Conditions. In: RABIN, S. **AI Game Programming Wisdom**. New York: Charles River Media. 2002. p.460-472.

AZEREDO, T.R., VIEIRA, P.S., MATIAS, I.O, VIANNA, D. S., Neto A.A.S. **Um controle autônomo de um carro de corrida.** V Simpósio Brasileiro de Games e Entretenimento Digital. 2006.

BIASILLO, G. Representing a Race Track for the AI. In: RABIN, S. **AI Game Programming Wisdom**. New York: Charles River Media. 2002a. p.439-443.

BIASILLO, G. Racing AI Logic. In: RABIN, S. **AI Game Programming Wisdom**. New York: Charles River Media. 2002b. p.444-454.

OGRE. Community Add-on Projects. 2000b. Disponível em http://www.ogre3d.org/index.php?option=com_content&task=view&id=17&Itemid=70. Acesso em 25 jul 2006.

SMITH, R. Open Dynamics Engine. 2006. Disponível em http://www.ode.org. Acesso em 26 jul 2006.