

Navegação em jogos eletrônicos usando PathFinding e Steering Behavior

Thiago Dias Pastor e Bruno Duarte Corrêa

Escola Politécnica da Universidade de São Paulo
thiagodiaspastor@gmail.com bruno.duarte@gmail.com
<http://www.poli.usp.br>

Resumo. Este paper apresenta duas soluções para navegação em mundos virtuais, a primeira baseada em Inteligência Artificial usando o famoso algoritmo A*, a segunda baseada na teoria de campos elétricos conhecida como Steering Behavior. O artigo explica o funcionamento de cada uma das técnicas, suas vantagens e desvantagens, além de mostrar alguns exemplos. Ao fim é proposta uma abordagem híbrida utilizando as duas soluções.

Palavras-chave: Inteligência Artificial, Steering Behavior, PathFinding, A*, Navegação, Jogos Eletrônicos

1 Introdução

O avanço do hardware possibilita ao desenvolvedor de jogos eletrônicos aproximar-se cada vez mais do mundo real. A maioria dos jogos, do ponto de vista do desenvolvedor, se baseiam em três pilares fundamentais, a computação gráfica, a simulação física e a inteligência artificial. Com o surgimento de placas de vídeos cada vez mais potentes os pilares relacionados com computação gráfica e simulação física estão migrando para a GPU deixando a CPU quase que exclusivamente dedicada à computação de questões relacionadas com a inteligência artificial.

Neste contexto, os sistemas de navegação em mundos virtuais ganham destaque, jogos de estratégia em tempo real e FPS (First Person Shooter) são exemplos de jogos em que a experiência do jogador depende bastante da eficiência e realismo deste tipo de sistema.

Um importante requisito envolvido nestes sistemas é tentar simular, o mais próximo possível do real, o comportamento que um humano teria se colocado nas mesmas situações em que os agentes de um jogo estão submetidos, não bastos ter a resposta correta em um tempo muito pequeno. A maioria dos mundos virtuais são dinâmicos e a navegação deve ter a capacidade de se adaptar a diversas alterações de parâmetros em tempo real.

Um desafio a mais para estes sistemas é a questão da integração com os outros módulos do jogo. As ações escolhidas pelo sistema de navegação devem ser enviadas para os sistema de animação de personagem e simulação física, e as respostas do sistema físico devem ser percebidas pelo sistema de navegação.

Este artigo descreve duas técnicas que são utilizadas em sistemas de navegação modernos para jogos eletrônicos. Por fim é sugerida uma abordagem híbrida envolvendo elementos das duas técnicas.

2 Sistemas de navegação em jogos

Um sistema de navegação para jogos eletrônicos consiste em um módulo específico responsável por planejar qual caminho um determinado agente deve seguir para atingir um objetivo específico. A movimentação dos agente, assim como desviar de obstáculos estáticos e dinâmicos fazem parte do escopo do sistema, porém a escolha de objetivos não o faz.

Um mundo virtual é constituído de diversos objetos como ruas, carros, personagens, etc. Cada elemento deste universo é representado por uma entidade física correspondente, por exemplo, um carro pode ser representado por uma caixa e um personagem por um cilindro. Todas estas entidades ocupam um volume no espaço virtual (duas entidades a priori, não podem ocupar o mesmo espaço, caso isto ocorra, a simulação física irá tratar esta situação como um choque entre entidades e o processará de acordo com a física newtoniana).

Um agente é um objeto virtual dinâmico deste mundo, ele possui uma velocidade e uma direção de movimento. Em alguns ambiente os agentes podem conter sensores através dos quais fazem inferências sobre o estado do mundo, normalmente os agentes têm acesso completo a estrutura topológica do mundo virtual.

Neste paper, foi considerado responsabilidade do sistema de navegação controlar ativamente as variáveis físicas de uma agente a fim de levá-lo de um local para outro do universo virtual, em algumas implementações utilizadas pelo mercado, existe uma sistema auxiliar responsável apenas pela movimentação dos agentes.

Ao longo dos anos duas abordagens para a construção deste tipo sistema se destacaram. A primeira é discretizar o mundo virtual em nós conhecidos como waypoints, os quais simbolizam áreas atingíveis pelos agentes. A cada waypoint associa-se um valor que descreve o quão difícil é para um agente ficar neste ponto. Em seguida adota-se uma política para conectá-los e em seguida um grafo de conexão é gerado. Por exemplo uma conexão entre waypoints A e B significa que um agente estando em A pode atingir B seguindo em linha reta. Quando um agente quiser ir de um ponto X para um ponto Y, basta procurar o waypoint mais próximo de X e o mais próximo de Y, em seguida, usando a teoria de grafos encontrar o caminho de menor custo entre eles. Por fim o agente se movimentará pelos waypoints até chegar ao seu destino. Um dos maiores problemas associados a esta abordagem é primeiramente a artificialidade dos caminhos gerados, outros problemas estão relacionados à dinamicidade do ambiente (o surgimento de um obstáculo requer uma recomputação (parcial) do algoritmo), ao alto custo para computação e a alta previsibilidade de comportamento dos agentes.

A segunda abordagem considera os elementos que bloqueiam o agente (paredes por exemplo) como um geradores de campo repulsor e o objetivo como

um gerador de campo atrator. A intensidade destes campos é proporcional a distância do agente ao elemento gerador dos mesmos. A sistemática de funcionamento é extremamente simples, a cada instante computa-se uma resultante entre a força atratora que o objetivo exerce sobre o agente e as forças repulsoras que os obstáculos geram, o agente deve se movimentar na direção da resultante. As maiores vantagens deste método estão relacionadas com a rapidez de cálculo e a capacidade de lidar com situações bastante dinâmicas, porém uma grande desvantagem é a possibilidade do agente ficar preso em alguns locais (existem situações em que os obstáculos podem gerar um campo que prende o agente em uma região do espaço).

Em seguida serão discutidas as duas abordagens detalhadamente.

2.1 Navegação com PathFinding A*

Conforme explicado anteriormente, esta abordagem necessita de um sistema de waypoints. Existem algumas maneiras de se construir estes elementos. As mais utilizadas no mercado são NavMesh, posicionamento Manual e posicionamento automático seguindo heurística.

A primeira técnica conhecida como NavMesh (modelo de Navegação) consiste na geração de waypoints no momento em que o designer está construindo o modelo visual dos mundo virtual (Ex: no momento em que o modelador esta criando uma cidade ele coloca informações sobre quais lugares desta cidade podem ser visitados por um agente e quais lugares não podem). O desenvolvedor deve processar este modelo, retirar estas informações de navegação e construir o grafo de waypoints a partir delas. Uma segunda maneira bastante usual é processar os triângulos de um modelo (não há necessidade do designer adicionar informações extras) calculando a inclinação entre vértices adjacentes, se o valor for maior que um determinado limite o personagem não conseguira atingir esta região, e não são gerados waypoints nestas posições, em seguida utiliza-se informações de visibilidade (para cada waypoint calcula-se quais outros são acessíveis por um agente caminhando em linha reta) para criar o grafo de waypoints.

A segunda maneira é através de posicionamento manual, no qual um designer posiciona manualmente os waypoints no mundo virtual (o designer controla manualmente um agente e posiciona cada um dos waypoints no mundo). A conexão entre eles é realizada utilizando informações de visibilidade automaticamente.

A ultima maneira é bastante semelhante à anterior, porém os waypoints são criados automaticamente, um sistema posiciona-os seguindo um padrão (um exemplo seria dividir o mundo em uma grid e colocar um waypoint em cada vértice de cada quadrado do grid). A conexão é feita novamente utilizando informações de visibilidade automaticamente. Este último método é completamente automático, porém é bastante ineficiente, uma vez que vários waypoints gerados não serão atingíveis, não haverá controle granulado sobre quais locais o agente poderá ir e haverá bastante redundância.

A técnica mais eficiente, por possibilitar um refinamento bastante preciso, é o NavMesh, apesar de requer que o modelo gráfico tenha um formato específico para conter as informações de navegação (empresas pequenas não têm como

desprender esforços para criar um formato específico para este fim, e a maioria dos formatos padrões não contem esse tipo de informação). Na impossibilidade de usar o NavMesh a abordagem mais adequada é adotar posicionamento manual, porém em mundos virtuais gigantesco esta abordagem é complicada e nestas situações costuma-se usar uma técnica híbrida que envolve o posicionamento automático com alguns refinamentos manuais.

Uma vez criado o grafo de waypoints, o próximo passo é a procura pelo caminho com o menor custo entre dois nós. O algoritmo mais utilizado é o A*. Existem diversas maneiras de implementá-lo. Utilizou-se a seguinte :

Lista aberta e Lista fechada são estruturas de dados referentes a listas de prioridades ordenadas por F (Fitness). G é o custo do movimento para se mover do ponto de início até o waypoint determinado na malha seguindo o caminho criado. H é o custo estimado do movimento entre o waypoint determinado e o destino final. Para o cálculo de H foi utilizado a heurística Manhattan descrita pela equação $|x_1 - x_2| + |y_1 - y_2|$. O Custo G é calculado utilizando os valores de custo armazenados nos waypoints. A estrutura de dados "waypoint" contém os valores de G, H, F e listas de filhos e pais.

1. Adicionar o waypoint inicial a uma lista aberta.
2. Repita o seguinte:
 - (a) Procure o waypoint que tenha o menor custo de F na lista aberta. Nós referimos a isto como o waypoint corrente.
 - (b) Mova-o para a lista fechada.
 - (c) Para cada um dos waypoints adjacente (Wa) a este waypoint corrente (Wc).
 - i. Se Wa não é atingível ou se estiver na lista fechada, ignore. Caso contrário faça o seguinte:
 - ii. Se Wa nao estiver na lista aberta, acrescente-o à lista aberta. Faça o Wc o pai de Wa. Grave os custos F, G, e H do waypoint.
 - iii. Se já estiver na lista aberta, conferir se o caminho atual para Wa é melhor, usando custo G como medida. Um valor G mais baixo mostra que este é um caminho melhor. Nesse caso, mudar o pai de Wa para o Wc, e recalcular os valores de G e F.
 - (d) Parar quando acrescentar o DESTINO a lista fechada (encontrou !!!) ou a lista aberta ficar vazia (não encontrou !!!)
3. O caminho a ser percorrido sera encontrado a partir dos pais de DESTINO até encontrar INICIO.

Uma vez encontrado o caminho a seguir, deve-se controlar a movimentação do agente. A maneira mais simples é utilizar os waypoints como entrada para uma Spline e a cada instante posicionar o agente em um ponto da curva gerada. Este procedimento suaviza o caminho gerado tentando dar uma naturalidade maior à movimentação. Outras abordagens, como calcular o diferencial de velocidade que deve ser adicionado ao agente para ele se alinhar com o próximo waypoint, podem ser adotadas.

Conforme discutido, um problema em potencial é a existência de objetos dinâmicos. Nesta técnica eles são completamente desprezados, pode-se pensar

em alterar os valores dos pesos dos waypoints (aumentar o custo dos ocupados) e recalcular o algoritmo, porém o processamento envolvido torna esta abordagem pouco prática em tempo real.

Normalmente, costuma-se colocar um limite entre as iterações do A* por frame, ou seja, a cada iteração do jogo apenas um pedaço do caminho é calculado, uma vez que o caminho completo não é necessário no início. Esta abordagem diminui consideravelmente o processamento.

Para este paper foi implementado os sistemas de posicionamento manual e automático para criação e conexão de waypoints. Além disto foi implementada a versão do algoritmo A* acima explicada. A movimentação do agente foi feita usando uma Bezier cujos pontos de controle são os waypoints que compõe o caminho. Todo o código foi escrito em C# e XNA usando a Engine PloobsEngine também desenvolvida pelos autores deste artigo.

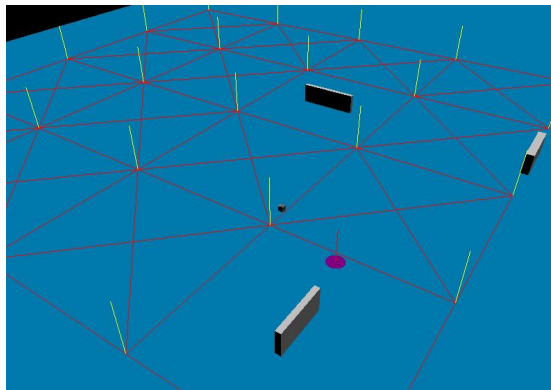


Fig. 1. Demo A* e Waypoints

Steering Behaviors

Steering Behaviors é um tópico bastante grande e não restrito à navegação, diversos efeitos interessantes como Flocking podem ser obtidos com o uso desta técnica. Neste paper nos restringimos a Steering Behaviors aplicado a navegação, em especial aos behaviors : AvoidObstacle, Seek, StayOnPath e AvoidNeighbors. Existem diversas maneiras de enxergar os Steering Behaviors, foi escolhida a abordagem baseada em campos elétricos pela sua fácil compreensão.

Esta segunda abordagem é bem mais simples que a anterior e consegue resultados bastante naturais. Conforme dito, cada objeto no mundo virtual cria um campo atrator ou repulsor. (Conceito bastante semelhante aos campos elétricos). Se um agente estiver sob o raio de ação de um destes campos, uma força proporcional ao valor dele irá aparecer no agente. A intensidade e o decaimento destes

campos são parâmetros de difícil medição cujos valores vêm da experiência do designer e da experimentação.

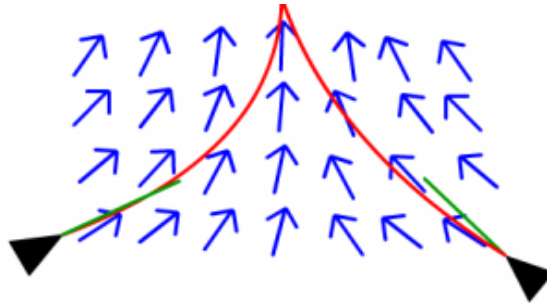


Fig. 2. Campos sobre um agente

O projeto de um sistema de navegação seguindo Steering behaviors é completamente diferente dos sistemas clássicos usando PathFinding. Não existe fase de pré-processamento para geração de waypoints além da diferença na semântica das variáveis que neste caso estão sempre relacionadas com a física.

O primeiro passo de um sistema de Steering Behaviors é definir um agente (o agente de Steering contém alguns atributos a mais do que um agente de PathFinding) que normalmente é modelado como um Veículo (entidade que possui uma aceleração, um torque e uma velocidade máximas, além de um vetor direção que aponta para a sua frente e um sistema auxiliar que irá converter o resultante que o agente está submetido em variáveis físicas como aceleração). Este agente irá possuir alguns comportamentos chamados na técnica de behaviors. Um comportamento descreve como o agente irá enxergar os campos do mundo virtual (atrator ou repulsor). A seguir os comportamentos usados pelo sistema de navegação serão descritos:

AvoidObstacle Comportamento bastante simples que consiste em enxergar os campos dos obstáculos estáticos como repulsores que decaem com a distância. Não ha parâmetros de entrada

Seek Comportamento que consiste em enxergar o Destino como um atrator, ele será um campo uniforme e não decairá com a distância. O Parâmetro de entrada é a posição do destino

AvoidNeighbors Comportamento bastante complexo (sua explicação detalhada esta fora do escopo do artigo) que consiste em evitar colisões com outros agentes. O método usado é baseado em previsão futura de posição (supor que os agentes manterão a mesma aceleração e velocidade, e prever sua posição em um tempo futuro próximo, verificar se existe alguma colisão em potencial e se houver enxergar a posicao da colisão como campo repulsor – Ver imagem abaixo). Não ha parâmetros de entrada relevantes.

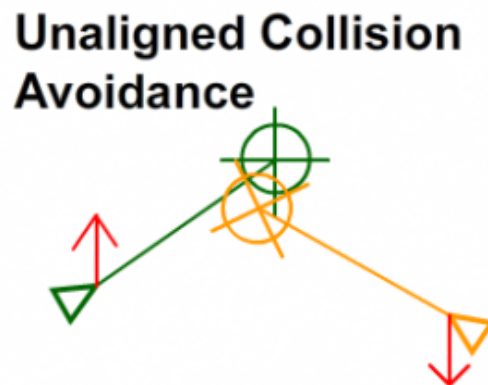


Fig. 3. Behavior AvoidNeighbors

Cada agente conterà os três behaviors citados acima. Eles são naturalmente combináveis uma vez que cada behavior produzira uma força (vetor) como saída. Existem algumas maneiras de realizar esta combinação, usaremos uma media ponderada simples:

$$\text{Resultante} = \text{AvoidNeighbors} * P_{an} + \text{AvoidObstacle} * P_{av} + \text{Seek} * P_s$$

$$P_{an} + P_{av} + P_s = 1$$

As variáveis P_{an} , P_{av} e P_s são frações que representam o quanto cada um dos behaviors representará na resultante final. Estes valores são obtidos através de experimentação.

A cada instante de tempo, os três behaviors são avaliados e combinados resultando em uma única força, em seguida ela é convertida em variáveis de movimentação (aceleração, velocidade e por fim em posição final) usando as equações newtonianas de movimento e por fim aplicadas ao agente.

Para este paper foi utilizada a implementação de Steering Behaviors da PloobsEngine (desenvolvida pelos autores deste artigo).

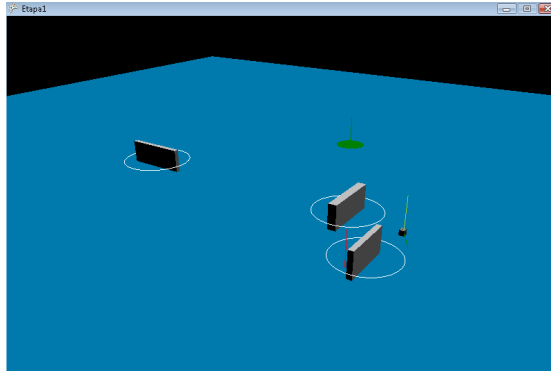


Fig. 4. Demo Steer Behaviors

Existem algumas implementações como a opensource OpeenSteer em C++.

A maior desvantagem deste método é o não determinismo, por exemplo em alguns momentos o agente fique preso em alguma região. Existem algumas técnicas para detectar estas situações e corrigi-las, porém não é algo suficientemente estável para ser utilizado em jogos triple-A (jogos de grandes produtoras feitos para atingir uma grande fatia do mercado). Em contrapartida, a naturalidade de movimento e a simplicidade em termos de processamento são características que o tornam bastante atraente.

Abordagem Híbrida

A fim de combinar as vantagens dos dois métodos (observar que elas são inerentemente complementares) este artigo propõe uma abordagem híbrida entre as duas técnicas.

A ideia é usando PathFinding, gerar um caminho entre o início e o destino, em seguida, criar um behavior, chamado StayOnPath, cuja função seria manter o agente dentro dele. Simplificadamente, pode-se enxergar o caminho gerado pelo PathFinding como uma sequência de retas que o agente deve seguir. O behavior StayOnPath enxergaria esta sequência de retas como uma diretriz de movimento, ou seja, ao invés de criar um campo atrator no destino (usando o behavior Seek), cria-se um campo que tende a atrair o agente para o caminho gerado. O behavior StayOnPath enxerga este percurso como uma série de cilindros posicionados na direção das retas (o raio de cada cilindro seria uma medida de erro que controlaria o quanto o agente poderia se desviar) e gera forças na tentativa de manter o agente dentro dele. É interessante observar que se algum objeto dinâmico aparecer durante a navegação o agente irá evitá-lo, uma vez que o behavior AvoidNeighbors estará ativo (o quanto e quando ele irá desviar dependerá do valor dos pesos ajustados na combinação de behaviors).

Para este paper foi implementado o behavior StayOnPath extendendo a coleção de comportamentos da PloobsEngine.

3 Conclusões

Os sistemas de navegação são bastante importantes para o realismo e experiência dos jogadores. A evolução dos hardwares, em especial das placas de vídeo possibilitam sistemas de inteligência artificial cada vez mais poderosos e complexos, neste contexto, este paper apresentou duas técnicas normalmente utilizadas pelo mercado na construção de sistemas de navegação.

Os sistemas baseados em PathFinding têm como vantagem principal o controle (às vezes excessivo e muito previsível) sobre quais movimentos um agente irá tomar, porém é pouco natural. Em contrapartida, os sistemas baseados em Steering behavior são bastante naturais e adaptativos, entretanto têm-se pouco controle sobre quais caminhos de fato serão tomados.

O artigo propôs um sistema com o propósito de unir as duas abordagens. Este sistema é bem mais complexo em termos de processamento e de construção do que cada um separadamente, porém o resultado obtido compensam o esforço.

Por fim, o paper foi bastante generalista e não se aprofundou muito nas técnicas abordadas, alguns tópicos como escalabilidade destes algoritmos em mundos virtuais grandes (como é o caso do World Of Warcraft) e técnicas de otimizações não foram sequer mencionadas. Para maiores detalhes consultar a bibliografia.

Referências

1. S. Russel and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, USA. 2nd. Edition, (2003).
2. M. Buckland. AI Techniques for Game Programming (The Premier Press Game Development Series). Jones And Bartlett Publishers, 1st. Edition, (2002).
3. M. Buckland. Programming Game AI by Example. Jones And Bartlett Publishers, 1st. Edition, (2004).
4. C. W. Reynolds. Steering Behaviors For Autonomous Characters, (1997).
5. http://www.policyalmanac.org/games/aStarTutorial_port.htm. A * Pathfinding para Iniciantes, (2004).
6. <http://www.red3d.com/cwr/steer/>. Applets sobre Steering Behaviors, (2004).
7. <http://opensteer.sourceforge.net/>. Implementacao em C++ dos Steering Behaviors clássicos, (2004).
8. <http://aigamedev.com/>. Portal IA em Jogos, (2010).
9. <http://www.gamedev.net/>. Portal Desenvolvedores de jogos, (2010).
10. <http://www.ploobs.com.br/>. Ploobs Engine WebSite - Engine gráfica 3D para criação de mundos virtuais - Thiago Dias Pastor e Bruno Duarte Corrêa, (2010).