# Aplicação do algoritmo PRM para planejamento em jogos FPS

Renato Luiz de Freitas Cunha
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

## Abstract

In this paper, we present an implementation of the Probabilistic Roadmaps algorithm, a sampling based algorithm created for planning in the robots, in a First-Person Shooter (FPS) game. In our implementation, we use both random and quasi-random sampling techniques to solve the problem of path planning in the AssaultCube game.

**Keywords:** First person shooters, Probabilistic Roadmaps, Artificial Intelligence, AssaultCube, Bots, NPC, Hammersley point set

## Resumo

Neste artigo é apresentada uma implementação do algoritmo PRM (Probabilistic Roadmaps), um algoritmo de planejamento baseado em amostragem, em um jogo de ação em primeira pessoa (FPS). São usados tanto métodos de amostragem aleatória e quase-aleatória para o pré-processamento de mapas no jogo AssaultCube.

**Palavras-chave:** First person shooters, Probabilistic Roadmaps, Artificial Intelligence, AssaultCube, Bots, NPC, Conjunto de pontos de Hammersley

**Contato do autor:**

renatocunha@acm.org

## 1 Introdução

Em jogos de um único jogador onda há ação ou estratégia envolvidos, normalmente o processo de construção de roadmaps e planejamento de caminhos é de suma importância, visto que uma implementação com baixo desempenho pode atrapalhar o bom desenvolvimento do jogo quando uma solução ruim criará agentes que se movimentam erraticamente, quebrando a sensação de fantasia esperada pelo jogador.

O objetivo deste artigo é investigar o impacto no desempenho e na qualidade da solução do uso do algoritmo PRM em jogos de ação em primeira pessoa. A importância deste tipo de estudo se encontra no fato de que a melhoria no desempenho sem perda de qualidade diminui o tempo de espera enfrentado pelo usuário, tornando sua interação com o jogo mais agradável. Este trabalho, portanto, implementa o algoritmo PRM em um jogo funcional e completo de código aberto.

Este trabalho está organizado da seguinte forma: na seção 2 são apresentados os principais trabalhos relacionados. A seção 3 apresenta os conceitos mais importantes usados neste artigo e apresenta uma análise assintótica de complexidade do algoritmo PRM. Na seção 4 é apresentada a metodologia seguida para realização deste trabalho. A seção 5 apresenta detalhes da implementação das técnicas descritas neste trabalho. Os experimentos e resultados realizados são descritos na seção 6. Este trabalho é finalizado pela seção 7, que apresenta as conclusões sobre este trabalho e possíveis trabalhos futuros.

## 2 Trabalhos relacionados

Atualmente, existem diversos trabalhos implementando algoritmos da robótica em jogos eletrônicos, aproximando essas duas áreas. Em [Souza e Chaimowicz 2007] é usado o algoritmo RRT para randomização dos caminhos gerados pelos bots, com a contribuição de geração de caminhos aleatórios pelos bots, diminuindo a previsibilidade do planejador de caminhos.

[Orkin 2005] trata da arquitetura de agentes e considerações para que eles planejem tem tempo real de maneira satisfatória. As considerações de arquitetura são baseadas na arquitetura do jogo F.E.A.R.

Sobre planejamentos em jogos em geral, [Hagelbäck e Johansson 2008] trata sobre o uso de campos de potencial (outra técnica da robótica) em jogos de estratégia em tempo real. Nesse trabalho são apresentados os requisitos para um bom funcionamento e a implementação, com resultados e trabalhos futuros.

Sobre o algoritmo PRM, há o trabalho [Sánchez et al. 2002], onde são estudadas as aplicações de algoritmos quase-aleatórios ao PRM, bem como o uso de Lazy-PRM para o planejamento de robôs. Os ganhos de desempenho são interessantes e dignos de serem estudados em jogos. Sobre o uso específico de PRM em jogos, [Nieuwenhuisen et al. 2007] combina PRM com amostragem em diagramas de Voronoi para geração de roadmaps suaves, removendo a necessidade de suavização dos caminhos dos bots em uma etapa de pós-processamento.

Sobre o planejamento de robôs, há ainda [Atramentov e LaValle 2002], que estende o algoritmo ANN, baseado em Kd-Trees, e apresenta um algoritmo eficiente para realização de consultas de vizinhos mais próximos. A validação do algoritmo é realizada usando-o em conjunto com os algoritmos PRM e RRT.

## 3 Conceitos

Nesta seção são apresentados alguns conceitos usados neste artigo.

### 3.1 Grafo de navegação

Um grafo de navegação é um grafo $G = (V, E)$ que representa todas as localidades alcançáveis por agentes através do ambiente de um jogo. Aos vértices deste grafo normalmente se dá o nome de waypoints. Comumente, às arestas deste grafo são atribuídos pesos representando a distância euclidiana entre os waypoints [Buckland 2005].

### 3.2 Bot

Um bot é um personagem controlado pelo computador com o objetivo de simular um jogador humano em jogos FPS. Os bots são funcionam através de rotinas de inteligência artificial pré-programados para atender ao requisitos do jogo, como regras, mapas e outros parâmetros inerentes ao jogo. Neste trabalho, para navegação nos mapas, os bots de utilizam de grafos de navegação.

### 3.3 PRM

O método Probabilistic Roadmap (PRM) é um algoritmo de planejamento da robótica, que resolve o problema de encontrar um caminho entre uma configuração inicial de um robô e uma configuração objetivo evitando colisões. Este método pode ser usado em jogos como uma alternativa ao método automático de criação de grafos de navegação em jogos eletrônicos, como o representado na figura 1.

Sua idéia básica é a de obter amostras aleatórias do espaço de configuração do robô, testar se estar amostras são livres de colisões, adicioná-las ao grafo de navegação caso estejam livres e usar um algoritmo de busca em grafos para determinar um caminho entre as configurações objetivo e inicial. O algoritmo 1 apresenta a etapa de geração do grafo de navegação do PRM. Na linha 12 do algoritmo, a chamada $\Delta(q, q')$ retorna ou um caminho sem colisões de $q$ para

$q'$ ou NIL, caso este caminho não possa ser encontrado. Sua análise de complexidade é realizada na seção 3.3.1.

A etapa de resolução de uma consulta se encontra fora do escopo deste artigo. No entanto, [Choset et al. 2005] apresenta um estudo detalhado sobre o PRM e a fase resolução de consultas.

O algoritmo PRM é provado como probabilisticamente completo, o que quer dizer que, quando o número de pontos amostrados tende ao infinito, a probabilidade de o algoritmo não encontrar um caminho, se ele existir, tende a zero.

**Algoritmo 1** Algoritmo de Construção de Roadmap

**Entrada:** $n \wedge k$
**Saída:** A roadmap $G = (V, E)$
1: $V \leftarrow \emptyset, E \leftarrow \emptyset$
2: **while** $|V| < n$ **do**
3: **repeat**
4: $q \leftarrow$ a random configuration in $Q$
5: **until** $q$ is collision-free
6: $V \leftarrow V \cup \{q\}$
7: **end while**
8: **for all** $q \in V$ **do**
9: $N_q \leftarrow$ the $k$ closest neighbors of $q$
10: **for all** $q' \in N_q$ **do**
11: **if** $(q, q') \notin E \wedge \Delta(q, q') \neq$ NIL **then**
12: $E \leftarrow E \cup \{(q, q')\}$
13: **end if**
14: **end for**
15: **end for**

### 3.3.1 Análise de Complexidade do PRM

*A priori*, não é possível obter a complexidade do algoritmo de construção de roadmap do PRM devido a seu caráter não-determinístico. Desta forma, precisamos assumir algumas características para uma análise preliminar da complexidade deste algoritmo.

Assumindo, portanto, que a escolha de uma configuração aleatória pode ser feita em tempo $\Theta(1)$ e que esta escolha estará livre de colisões, podemos perceber que as linhas 3 a 7 do algoritmo 1 são executadas $|V|$ vezes. Logo, este trecho de código executará em tempo $\Theta(n)$. Para a linha 10, assumindo que a distância entre dois pontos é obtida em tempo $\Theta(1)$, a escolha dos $k$ vizinhos mais próximos de $q$ é realizada em tempo $O(k)$. Assumindo que todas as chamadas a $\Delta(q, q')$ retornam um caminho sem colisões e que nenhum dos caminhos $(q, q')$ testados existam previamente em $E$, o que constitui o pior caso do algoritmo pois gerará o maior conjunto $E$ possível, teremos o loop do trecho das linhas 11 a 15 executado $k$ vezes. Desta forma, o trecho das linhas 9 a 16 possui complexidade de tempo $O(kn)$. A complexidade total para o pior caso do algoritmo é dada por $O(n) + O(kn) = O(kn)$.

## 3.4 Conjunto de pontos de Hammersley

O conjunto de pontos de Hammersley, um conjunto determinístico (quase-aleatório) de baixa discrepância, pode ser combinado ao algoritmo PRM para obter pontos distribuídos de maneira mais uniforme no espaço [Choset et al. 2005].

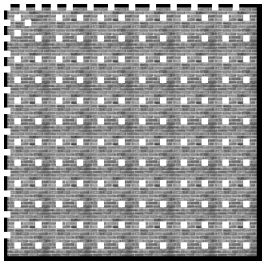

**Figura 1:** *Waypoints de um grafo de navegação baseado em grid.*

Como supracitado, sua aplicação pode gerar pontos no espaço mais bem distribuídos que uma função de escolha aleatória. A figura 2 exibe waypoints gerados por um método aleatório e pelo conjunto de pontos de Hammersley.

Este conjunto se baseia no fato de que todo inteiro não negativo $k$ pode ser expandido usando uma base de número primo $p$ como

$$k = a_0 + a_1 p + a_2 p^2 + \ldots + a_r p^r \tag{1}$$

Onde cada $a_i$ é um inteiro em $[0, p-1]$. É possível, então, definir uma função $\Phi_p(k)$ como

$$\Phi_p(k) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \ldots + \frac{a_r}{p^{r+1}} \tag{2}$$

Seja $d$ a dimensão de um espaço a ser amostrado. Qualquer seqüência $p_1, p_2, \ldots, p_{d-1}$ de números primos descreve a seqüência de funções $\Phi_{p1}, \Phi_{p2}, \ldots \Phi_{pd-1}$, onde o $k$-ésimo ponto $d$-dimensional de Hammersley é definido por

$$\left(\frac{k}{n}, \Phi_{p1}(k), \Phi_{p2}(k), \ldots, \Phi_{pd-1}(k)\right) \tag{3}$$

para $k = 0, 1, 2, \ldots, n-1$, onde $p_1 < p_2 < \ldots < p_{d-1}$ e $n$ é o número total de pontos de Hammersley. Para avaliação da função $\Phi_p(k)$, a função de complexidade para o pior caso é $O(\log_p k)$ para avaliação do $k$-ésimo ponto. A complexidade para a geração de um conjunto de $N$ pontos é $O(N \log_p N)$ [Wong et al. 1997].

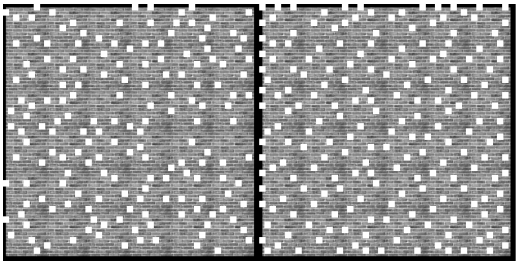

**Figura 2:** *Waypoints de um grafo de navegação baseado com amostragem aleatória (esquerda) e com conjunto de pontos de Hammersley (direita).*

## 3.5 AssaultCube

AssaultCube é um jogo FPS baseado na engine Cube que, por sua vez, é tanto um jogo FPS quanto uma engine para jogos FPS. Sua primeira versão oficial foi lançada em novembro de 2006, com o objetivo de criar um jogo totalmente novo em um ambiente mais realista. Sua temática do jogo é baseada nos conflitos entre grupos de operações especiais e grupos terroristas, popularizada pelo jogo Counter Strike, e, apesar de seu foco principal ser em modo de vários jogadores, ele conta com um modo monousuário que implementa a funcionalidade de uso de bots.

# 4 Metodologia

Para a realização do projeto descrito neste artigo, foram realizados os seguintes passos:

1. Revisão bibliográfica das técnicas de navegação usadas em robótica móvel potencialmente aplicáveis a jogos de FPS;
2. Determinação do problema a ser atacado (geração de waypoints);
3. Escolha de um jogo completo e funcional para validação dos algoritmos;
4. Implementação dos algoritmos;
5. Realização de experimentos comparando a nova implementação e a implementação tradicional;

6. Interpretação dos resultados obtidos.

No primeiro passo foram lidos diversos artigos descrevendo implementações de novos planejadores de caminhos em jogos e sobre algoritmos de planejamento da robótica. Daí, foi identificado um conjunto de algoritmos promissores, o de algoritmos baseados em amostragem e, então, o PRM.

Ao conhecer mais sobre o planejamento de caminhos em jogos, no segundo passo foi escolhida a parte de aprendizado do ambiente do jogo por ser uma parte normalmente custosa computacionalmente e que poderia ser beneficiada por algoritmos baseados em amostragem.

Para validação dos algoritmos implementados, foi definido que um jogo completo deveria ser usado. Dessa forma, não seria necessário implementar uma massa de dados sintética e o foco seria na implementação do algoritmo escolhido, sem a necessidade de preocupação com outros detalhes do jogo, como renderização, controle de entrada e saída e assim por diante. Dentre os jogos livres existentes, foi escolhido o jogo AssaultCube por já contar com uma implementação de bots que poderia ser facilmente modificada.

Os passos 4, 5 e 6 são descritos de maneira mais profunda nas seções 5 e 6.

# 5 Implementação

Conforme descrito anteriormente, o jogo AssaultCube conta com suporte a bots. Além de suporte a bots, ele conta com facilidades para edição de waypoints diretamente no jogo. São elas:

- Adição de waypoints e arestas entre waypoints através do uso de comandos de console de comandos embutido;
- Geração e destruição automática de waypoints através dos comandos *wpflood* e *wpclear*, respectivamente, do console do jogo;
- Salvamento e carregamento de arquivos de waypoints.

Como o objetivo deste artigo é o de estudar a geração automática de waypoints, a implementação se iniciou com a extensão da funcionalidade do comando *wpflood* e adição de novos comandos no console, como:

**setupstuff** Configura os parâmetros do jogo ativando o modo de um jogador com suporte a bots, adicionando-os e ativando a visualização de waypoint como entidades do jogo para fins de depuração;

**comehere** Ordena aos bots que planejem um caminho que inicia em suas posições iniciais e termina na posição atual do jogador;

**toggleprm** Comando que ativa ou desativa o uso de PRM para geração de waypoints no jogo;

**prmparams** Comando que define os parâmetros da implementação do algoritmo PRM, como número $n$ de pontos, $k$ vizinhos e se deve ser usado o conjunto de Hammersley ou a amostragem aleatória.

Para facilitar os testes, a máquina de estados dos bots foi modificada para que eles não se movessem. Com este artifício, vários testes puderam ser realizados sem que os bots mudassem de posição, mantendo o tamanho do caminha calculado fixo e possibilitando a medição de um tempo médio para planejamento de caminhos.

Outra adição ao código foi a modificação do minimapa do jogo, habilitando a visualização dos waypoints sobre o mapa do jogo. Ao minimapa também foi adicionada a capacidade de observar os caminhos calculados pelos bots, permitindo a comparação visual de diversos caminhos gerados.

Durante a fase de implementação e testes iniciais do código, foi criado um mapa pequeno, simples e sem mudanças de nível de altura, onde era validado o estado do jogo. Após a conclusão da implementação, os testes passaram a ser realizados em mapas oficiais do jogo.

```
map = packages/maps/ac_mines.cgz

method = random
waypoint_vertices = 700
waypoint_time = 1259
```

**Figura 3:** *Dados sobre uma geração de 700 waypoints para o mapa ac_mines com duração de 1259 ms.*

```
astar_from = (98.000000, 71.000000, -2.000000)
astar_to = (120.000000, 223.000000, 4.000000)
astar_length = 231.618140
astar_time = 7.440000
astar_nodes = 47
```

**Figura 4:** *Dados sobre uma consulta de geração de caminho do ponto* $(98, 71, -2)$ *para o ponto* $(120, 223, 4)$ *com tamanho total de* $231.618140$*,* $47$ *waypoints e durando* $7.44$ *ms.*

Todo o código relacionado à geração do grafo de navegação e à resolução de consultas de caminho foi instrumentado para exibição de estatísticas sobre essas fases. As estatísticas geradas foram posteriormente coletadas e utilizadas para geração de gráficos e análise de desempenho da implementação. A figura 3 exibe um exemplo de dados sobre a fase de criação do grafo de navegação e a figura 4 exibe dados sobre uma consulta específica.

# 6 Experimentos

O seguinte conjunto de experimentos foi definido para validação da implementação:

- Tempo médio gasto para geração de grafo de navegação para o mapa, exibido na figura 5;
- Tempo médio gasto para geração de caminhos usando os métodos de Hammersley e aleatório variando o número de vértices usados 6;
- Tamanho médio do caminho gerado usando diferentes métodos de geração de caminhos, exibido na figura 7;
- Variação do tamanho do caminho gerado pelo método de amostragem aleatório, exibido na figura 8;

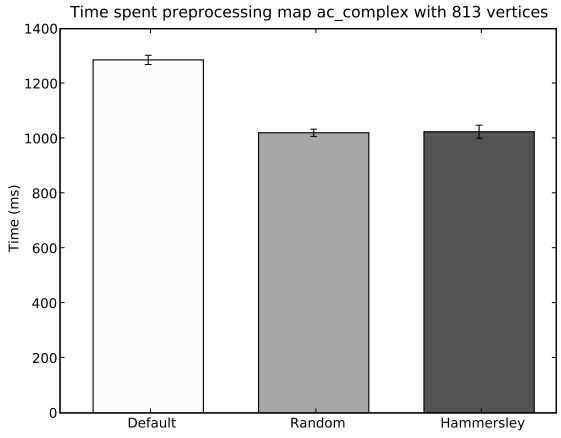


**Figura 5:** *Comparação entre os tempos de pré-processamento para 813 waypoints no mapa ac_complex para os métodos padrão, aleatório e Hammersley.*

Os experimentos exibidos nas figuras 6 e 7 possuem como posição inicial o ponto $(82, 78, -2)$ e, como destino, o ponto $(120, 223, 4)$. Esses pontos foram escolhidos pois se situam em posições praticamente opostas no mapa e entre eles há ramificações que fazer com que o caminho planejado varie de acordo com o posicionamento dos vértices do grafo de navegação. A figura 9 exibe um caminho gerado baseado nesses pontos.

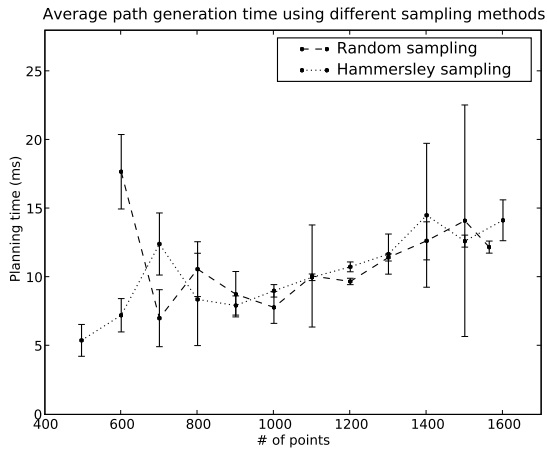


**Figura 6:** *Comparação entre os tempos de geração de caminho variando o número de vértices no grafo de navegação para o mapa ac_complex.*

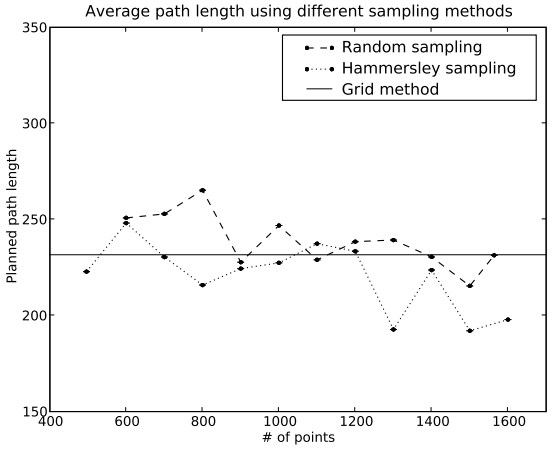


**Figura 7:** *Comparação entre os tamanhos de caminho gerados variando o número de vértices no grafo de navegação para o mapa ac_complex.*

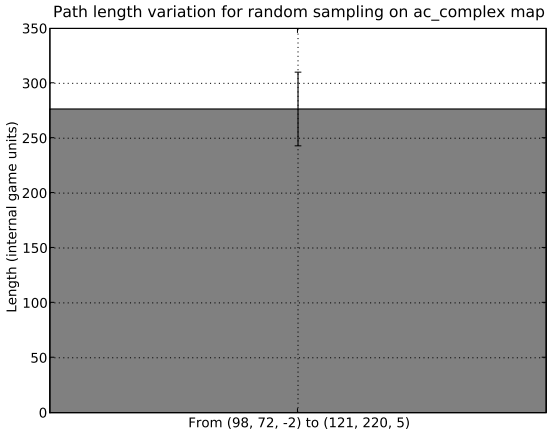


**Figura 8:** *Variação do tamanho do caminho gerado para o método de amostragem aleatório usando o mesmo número de pontos no grafo de navegação, mas variando as posições dos diversos vértices no mapa ac_complex.*

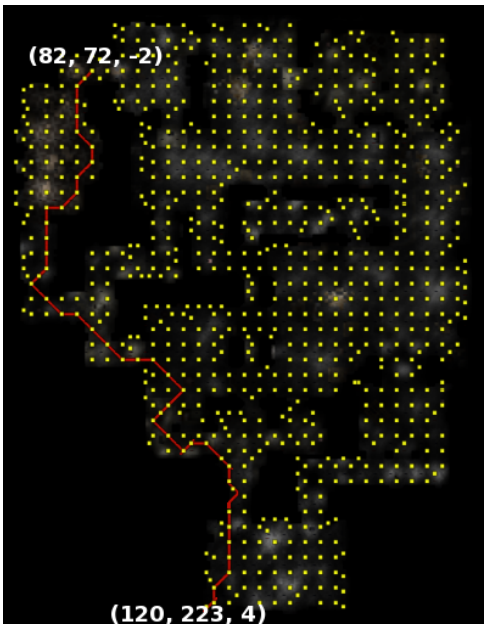


**Figura 9:** *Exemplo de caminho gerado sobre o mapa ac_complex. Neste mapa, o caminho exibido foi computado partindo do ponto* $(82, 78, -2)$ *para o ponto* $(120, 223, 4)$ *sobre um grafo de navegação gerado pelo método padrão baseado em grid.*

## 7 Conclusões e trabalhos futuros

Ao analisar os gráficos de desempenho da implementação (figuras 8 e 5), percebe-se que a aplicação do PRM ao planejamento em jogos melhora o desempenho da etapa de pré-processamento do mapa sem comprometer a qualidade dos caminhos gerados pelo planejador de caminhos.

Os resultados obtidos sugerem que o uso de PRM pode ser promissor. No entanto, a qualidade da solução não difere muito da gerada pela implementação padrão. Para ter certeza se o PRM pode ser aplicado com sucesso em jogos, seria melhor testá-lo em um ambiente mais dinâmico que o cenário de jogos FPS. Em jogos desse tipo, espera-se que algoritmos baseados em amostragem se comportem com desempenho superior à implementação padrão.

Para melhoria da qualidade da solução deste trabalho, é possível investigar os resultados obtidos por outras variantes do PRM, como o Lazy PRM [Sánchez et al. 2002], que adia a etapa de detecção de colisão para a fase de resolução de consultas. Desta forma, a etapa de pré-processamento retornaria em pouco tempo e o custo da verificação das colisões seria amortizado pelas consultas para resolução de caminhos.

Outra alternativa para o trabalho pode ser o uso do conjunto de pontos de Halton[Wong et al. 1997] como forma de reduzir o tempo de pré-processamento gasto. Na implementação atual, uma possível fonte de atraso na geração dos waypoints é a grande taxa de colisões encontradas ao usar os pontos de Hammersley, devido a sua distribuição uniforme. Como nessa seqüência o número do pontos a serem gerados deve ser conhecida *a priori*, para gerar um número $n$ de pontos em um ambiente com obstáculos, comumente é necessário usar como argumento da função de geração de pontos um valor $m \gg n$, resultando em um maior ainda número de colisões. Ao usar uma seqüência como a de Halton, onde os pontos do conjunto podem ser gerados sob demanda, é possível que o número de colisões seja menor e, conseqüentemente, que o tempo de pré-processamento diminua. De qualquer forma, uma nova implementação deve ser feita para verificar a validade dessa afirmação.

## Referências

ATRAMENTOV, A., E LAVALLE, S. M. 2002. Efficient nearest neighbor searching for mot ion planning. Em *Proceedings of the*

*International Conference on Robotics and Automation*, 632–637.

BUCKLAND, M. 2005. *Programming Game AI by Example*. Wordware.

CHOSET, H., LYNCH, K. ., HUTCHINSON, S., KANTOR, G., BURGARD, W., KAVRAKI, L. E., E THRUN, S. 2005. *Principles of Robot Motion: Theory, algorithms and implementations*. MIT Press.

HAGELBÄCK, J., E JOHANSSON, S. 2008. Agent architecture considerations for real-time planning in games. Em *Proceedings of 7th Int. Conf. on Autonomous Agents and Multiagent Systems*, 631–635.

NIEUWENHUISEN, D., KAMPHUIS, A., E OVERMARS, M. 2007. High quality navigation in computer games. *Science of Computer Programming 67*, 1 (June), 91–104.

ORKIN, J. 2005. Agent architecture considerations for real-time planning in games. Em *Proceedings of AIIDE*.

SOUZA, S. A., E CHAIMOWICZ, L. 2007. Utilizando rapidly-exploring random trees (rrts) para o planejamento de caminhos em jogos. Em *Proceedings of the VI Brazilian Symposium on Computer Games and Digital Entertainment*, 170–177.

SÁNCHEZ, A., ARENAS, J. A., E ZAPATA, R. 2002. Non-holonomic path planning using a quasi-random prm approach. Em *Proceedings of the International Conference on Intelligent Robots and Systems*, 2305–2310.

WONG, T.-T., LUK, W.-S., E HENG, P.-A. 1997. Sampling with hammersley and halton points. *J. Graph. Tools 2*, 2, 9–24.