UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOSÉ ANTÔNIO SALINI FERREIRA

Implementação e Teste de um Algoritmo Planejador de Caminhos em um Jogo de Estratégia de Tempo Real

Trabalho de Graduação

Profª. Drª. Luciana P. Nedel
Orientador

Porto Alegre, novembro de 2010.

AGRADECIMENTOS

Os seguintes amigos contribuíram em menor ou maior grau para a realização deste trabalho, e merecem minha gratidão:

À Dona Teresinha, minha mãe e principal apoiadora na minha caminhada pela UFRGS, obrigado por todo o carinho, apoio incondicional e compreensão sempre.

À Thais, namorada e sempre companheira, que me apoiou e incentivou na realização deste trabalho, e que me trata sempre com tanto carinho e atenção.

À todos meus colegas de curso e amigos da faculdade, especialmente Caesar, Filipe, e Thiago, a qual tivemos muitas discussões técnicas e debates à respeito de informática e tecnologia.

Aos meus amigos mais próximos, especialmente Bruno, Márcio, e Vinícius, que foram compreensivos com o meu isolamento nos últimos dois semestres da faculdade, e mesmo assim ainda continuaram lembrando de mim e me convidando para os churrascos e demais encontros.

Aos colegas do serviço, pelo apoio na realização deste trabalho.

À equipe do Grupo de Computação Gráfica da UFRGS, especialmente minha orientadora, Profa. Dra. Luciana Nedel, pela liberdade, confiança e incentivo depositados, e aos colegas Renato e Leonardo pelos esclarecimentos técnicos e ajuda fornecida.

Para finalizar, agradeço ao corpo de professores e funcionários da UFRGS, pela excelente estrutura de curso fornecida. Desde a biblioteca organizada e bem suprida, até as salas de aula e laboratórios bem equipados, cada detalhe do curso foi especialmente projetado para favorecer a criação de um ambiente acadêmico de qualidade.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

RTS	*Real time strategy*
SMA	Sistema multiagente
BVP	*Boundary value problem*
GPU	*Graphics processing unit*
GPL	*General Public License*
IA	Inteligência artifical
DLL	*Dynamic-link library*
API	*Application programming interface*
OpenGL	*Open Graphics Library*
FPS	*First-person shooter* ou *Frames*-por-segundo
MMORPG	*Massive Multiplayer Online Role-Playing Game*
RPG	*Role-Playing Game*
SDL	*Simple DirectMedia Layer*
PNG	*Portable Network Graphics*
SVN	*Subversion*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O planejamento de caminhos faz parte dos principais módulos de um sistema multiagente com ambientes dinâmicos, e tem como objetivo ajudar a garantir um nível mínimo de autonomia de movimentação aos agentes. Simulações contendo sistemas multiagente em ambientes dinâmicos são muito empregadas em jogos eletrônicos, e com o crescimento do segmento comercial de jogos, incluindo jogos para dispositivos portáteis e celulares, as tecnologias empregadas em sistemas multiagentes ficaram em evidência, a fim de garantir os diversos níveis de detalhe exigidos pelo público usuário da simulação. O objetivo deste trabalho foi realizar a implementação e análise de um planejador de caminhos que usa campos potenciais em um jogo de estratégia em tempo real (RTS), de forma a permitir uma comparação com outro planejador de caminhos, e verificação da qualidade e demais características do método proposto.

Durante o trabalho, são apresentadas as informações técnicas sobre o planejador de caminhos usado, além de dados à respeito do jogo utilizado para os testes, e os detalhes de integração referentes à implementação do novo algoritmo no jogo. Também são descritos os testes realizados, que envolveram verificações de desempenho e qualidade, além de um teste de vantagem estratégica. Os testes de desempenho mostram a capacidade de escalabilidade e limites gerais do algoritmo planejador de caminhos novo em comparação ao planejador de caminhos de referência. Os testes de qualidade mostram os caminhos gerados pelos dois planejadores em diversas situações diferentes, permitindo que seja feita uma comparação em termos de naturalidade dos caminhos. Por fim, os testes de vantagem estratégica apresentam resultados de combates entre os dois planejadores distintos, com o objetivo de apurar a possível existência de vantagens ou desvantagens entre o uso de cada método.

Palavras-chave: Planejamento de caminho, jogo de estratégia em tempo real, campo potencial, RTS.

Implementation and test of a path planning algorithm in a real time strategy game

ABSTRACT

The path planning is part of the main modules of a multi-agent system with dynamic environments, and aims to help ensure a minimum level of autonomy to mobile agents. Multi-agent simulations in dynamic environments are widely used in electronic games, and accompanying the growth of commercial games industry, including games for hand-held devices and mobile phones, the technology of multi-agent system gains importance, in order to ensure the various levels of detail required by the public and users of simulations. The objective of this work is the implementation and analysis of a path planner which uses potential fields in a real time strategy (RTS) game, to allow a comparison with another path planner, and checking of quality and other characteristics of the proposed method.

During the work, we present the technical information about the path planner used, along with general data about the game used for testing, and integration details concerning the implementation of the new algorithm in the RTS game. We also describe the tests that are involved, which are checks of performance and quality, plus a test of strategic advantage. Performance tests show the scalability of the method, along with general limits of the new path planner algorithm in comparison to the reference path planner. The quality tests show the paths generated by the two planners in several different situations in game, allowing a comparison in terms of the naturalness of the paths. Finally, tests of strategic advantage presented results of battles between the two different path planners, with the aim of investigating the possible existence of advantages or disadvantages on using each method.

Keywords: Path planning, real time strategy game, RTS, potential field.

1 INTRODUÇÃO

O segmento de jogos eletrônicos tem crescido muito nos últimos anos, ajudado também pela popularização dos jogos em dispositivos portáteis, e da mesma maneira, tem crescido a demanda do público da área por novas tecnologias, tanto de hardware como de software. Novas placas aceleradoras são desenvolvidas pelos fabricantes a cada dia, bem como diversos outros tipos de periféricos e dispositivos variados para satisfazer a vontade de todo tipo de jogadores, desde os mais sérios, até os jogadores casuais. Novos tipos de jogos são construídos com o objetivo de tentar agradar os gostos mais detalhistas dos jogadores, e o nível de detalhes e realismo das simulações nunca esteve tão alto.

1.1 Motivação

Alguns dos tipos de jogos mais populares, como os jogos RTS (*Real Time Strategy*), empregam o conceito de agente ou sistemas multiagente para realizar a simulação dos pormenores do jogo apresentado aos jogadores (PER 2005). Pela própria natureza da simulação, o ambiente de jogo tende a ser uma área bem definida, limitada, porém dinâmica, na medida em que os obstáculos e objetivos individuais de cada agente, e globais de cada jogador, podem estar se movendo ou mudando de estado durante o decorrer da simulação.

Conforme lembrado por Wooldridge (WOO 2002), em uma simulação de realidade virtual ou jogo de computador, espera-se que os agentes "forneçam a ilusão de vida, viabilizando a anulação de descrença da audiência"[1] (Bates *apud* WOO 2002). Ainda segundo Wooldridge, os agentes precisam mostrar emoção, agindo e reagindo de uma maneira razoável conforme nossa empatia e entendimento do comportamento humano (WOO 2002). Seguindo a tendência de exigência de mais realismo pelo público em geral das simulações eletrônicas e jogos, juntamente com a necessidade de agentes que ajam de uma maneira razoável e mais próxima do comportamento humano, é de se esperar que a área de jogos empregue algoritmos que, além de resolverem os problemas inerentes à simulação, forneçam um comportamento humano ou mais natural aos agentes da simulação, melhorando a qualidade geral da simulação e permitindo que o público perceba a "ilusão de vida" proposta por Wooldridge.

Em vista disso, o método de planejamento de caminhos descrito em Silveira *et al.* (SIL 2009), que propõe um comportamento natural de direção para pedestres virtuais, se mostra em sintonia com a necessidade de realismo dos agentes simulados em um jogo.

[1] Texto original: "*provide the illusion of life, thus permiting the audience's suspension of disbelief*" (Bates *et al.*, 1992, p.1).

1.2 Objetivo

Devido à característica de sistema multiagente de um jogo RTS, e à relevância do planejamento de caminhos dentro do jogo, o objetivo principal deste trabalho é a implementação em um jogo do tipo RTS do algoritmo de planejamento de caminhos proposto por Silveira *et al.* (SIL 2009), bem como a realização de testes experimentais para análise dos diversos fatores de qualidade e desempenho obtidos da utilização do algoritmo no jogo. O tipo de jogo foi escolhido pela natureza da simulação do jogo RTS, que possui um ambiente dinâmico que pode ser considerado um sistema de multiagentes, e onde a utilização do módulo de planejamento de caminhos proposto visa melhorar o comportamento dos agentes, de forma à obter-se um comportamento mais natural, ou até mesmo próximo do comportamento humano, no que diz respeito à movimentação dos indivíduos.

Os objetivos específicos deste trabalho envolvem:

- O estudo dos detalhes do método de planejamento de caminhos proposto por Silveira *et al.* (SIL 2009).
- A escolha de um jogo de RTS de código aberto.
- O estudo dos detalhes técnicos referentes ao planejamento de caminhos das unidades do jogo escolhido.
- Implementação e integração no jogo escolhido do novo método de planejamento de caminhos.
- Realização de testes de qualidade e velocidade, e verificação dos resultados encontrados.

Durante o transcurso deste trabalho, são apresentados alguns dos conceitos básicos das abstrações envolvidas, como agente, sistemas multiagentes, e ambientes dinâmicos. Também são apresentados alguns detalhes técnicos dos jogos de RTS em geral, bem como do jogo de código aberto usado nos testes, juntamente com o algoritmo de planejamento de caminhos implementado, e também do algoritmo nativo do jogo escolhido.

1.3 Conceitos importantes

Para a implementação do algoritmo planejador de caminhos no jogo de RTS, o conhecimento de alguns conceitos básicos é importante.

1.3.1 Sistema multiagente (SMA)

O termo "sistema multiagente" (SMA) aplica-se a sistemas com as seguintes características (FER 1999):

- Um ambiente, geralmente compreendido por um volume tridimensional onde se passa a simulação.
- Um conjunto de objetos, situados no ambiente. Os objetos são passivos, de forma que possam ser manipulados pelos agentes.
- Um conjunto de agentes, que representam os objetos ativos do sistema.

- Um conjunto de relações que ligam os objetos e agentes entre si.
- Um conjunto de operações, que permitem aos agentes perceberem, produzirem, consumirem, transformarem e manipularem objetos.
- Operadores, com o objetivo de representarem a aplicação das operações e a reação do ambiente em contrapartida à tentativa de modificação do mesmo, o que, em resumo, pode ser visto como "as leis do universo".

Uma ideia mais informal e vaga para sistemas multiagente poderia ser: um sistema multiagentes é implicitamente formado no momento em que uma simulação permita que vários agentes interajam em um ambiente comum, tanto colaborativamente, quando competitivamente.

1.3.1.1 Agente

Um agente é tudo o que pode ser considerado capaz de perceber seu **ambiente** por meio de **sensores** e de agir sobre esse ambiente por intermédio de atuadores (RUS 2004, p.33) [grifo do autor]. Agentes devem ser capazes de agir, e não apenas raciocinar (FER 1999, p.9). Um agente é um sistema de computação que é situado em um ambiente e que é capaz de ações autônomas no ambiente, a fim de fazer cumprir os seus objetivos primários (WOO 2002, p.15).

As três definições, apesar de pouco semelhantes, ajudam a fixar melhor a ideia vaga e ampla que um agente representa. Segundo Wooldridge, infelizmente não existe definição universalmente aceita para o termo agente, e ainda transcorre muito debate e controvérsia relativa ao assunto.

Contudo, a definição para agente será usada neste trabalho com o enfoque na área de jogos e, mais precisamente, no ambiente virtual simulado por um jogo, e sendo assim as três definições servem para situar a ideia do significado e do alcance de um agente. Para completar a definição de agente no contexto próprio deste trabalho, pode-se lembrar que a palavra agente vem do latim *agere*, e significa fazer. Sendo assim, o agente pode ser considerado simplesmente como algo que executa ações.

1.3.1.2 Ambiente

Os agentes devem estar situados em um ambiente (FER 1999), assim como o sistema multiagente deve conter, ou na verdade está contido ou se passa dentro de um ambiente.

Russell e Norvig (RUS 2004) ajudam a enumerar algumas propriedades interessantes dos ambientes, e que podem ser relevantes para a construção ou programação dos sistemas multiagentes ou dos agentes em particular:

- Completamente observável *versus* parcialmente observável: se os sensores dos agentes permitem acesso à todo o ambiente, ou somente parte é observável.
- Determinístico *versus* estocástico: o ambiente é determinístico se o próximo estado do ambiente é totalmente identificável pelo estado atual e pela ação executada pelo agente.
- Estático *versus* dinâmico: o ambiente é dinâmico se ele se altera enquanto o agente está deliberando sobre a tarefa a fazer em seguida. É considerado estático caso só se

altere com ações do próprio agente, ou não sofra alterações.

- Discreto *versus* contínuo: indica o tratamento dado aos estados possíveis do ambiente e às percepções dos agentes.
- Agente único *versus* multiagente: a ideia reside no tratamento dado à objetos e estruturas do ambiente, que podem ser estáticos, dinâmicos, ou podem até estar tentando maximizar seu desempenho inclusive reagindo às ações de outras estruturas ou agentes, ou até mesmo atuando de uma maneira cooperativa, ou mesmo competitiva.

1.3.2 Planejamento de caminhos

Os agentes inteligentes devem maximizar sua medida de desempenho (RUS 2004), e para atingir esse objetivo, pode estar envolvido algum tipo de movimentação espacial, caso o agente seja móvel.

Conforme Russell e Norvig, o problema do planejamento de caminho consiste em encontrar um caminho de uma configuração[2] até outra no espaço de configuração (RUS 2004). Sendo assim, o algoritmo de planejamento de caminhos pode ser visto como um meio de se fornecer uma mobilidade de movimento mínima para que os agentes possam transitar com sucesso entre pontos diferentes do ambiente dinâmico simulado, evitando ao máximo a colisão com obstáculos ou outros agentes, e possivelmente escolhendo os caminhos com menor custo de movimentação.

Considerando-se um ambiente simulado estático, onde os obstáculos e objetivos não variam, todas as rotas possíveis podem ser pré-calculadas. Entretanto, no momento em que as simulações começam a representar um ambiente com obstáculos dinâmicos, pode ser muito custoso computacionalmente pré-calcular todas as possibilidades de rotas levando em conta todas as combinações possíveis de situações e estados dos obstáculos e objetivos da simulação, e é nesse caso que o algoritmo de planejamento de caminhos se mostra realmente importante, já que permite fornecer a mobilidade dos agentes levando-se em conta a natureza variável do ambiente, o que contudo também não impede a utilização de planejamento de caminhos em ambientes estáticos.

1.3.2.1 Discretização, espaço livre e espaço ocupado

Segundo Russell e Norvig (RUS 2004), o espaço de configuração pode ser decomposto em dois subespaços: o espaço de todas as configurações que um robô[3] pode atingir, comumente chamado de **espaço livre**, e o espaço de configurações inacessíveis, chamado **espaço ocupado** [grifo do autor].

Em uma determinada abordagem de planejamento de caminhos, o espaço livre pode ser decomposto em um número finito de regiões contíguas, chamadas células, e o seu conjunto formando o *grid* de células. Quando isso ocorre, então o problema do planejamento de caminho se torna um problema de busca em grafo discreto, podendo ser resolvido de uma maneira semelhante aos problemas de busca clássicos (RUS 2004).

---

[2] O termo "espaço de configuração" é usado no sentido de "ambiente" do sistema multiagente.

[3] De acordo com Russell e Norvig, "os robôs são agentes físicos que executam tarefas manipulando o mundo físico".

1.3.2.2 Custo de movimentação

O custo de movimentação, ou custo de caminho, conforme definição de Russell e Norvig, visa atribuir um custo numérico a cada caminho, e dessa forma, o agente escolhe uma função de custo que reflete sua própria medida de desempenho. Russell e Norvig ainda supõem que o custo de um caminho pode ser descrito como a soma dos custos das ações individuais, ou custos de passo, ao longo do caminho. Conforme Perucia *et al.* (PER 2005), o custo determina a qualidade de um nodo, e quanto menor o custo, melhor a sua qualidade.

1.3.3 Jogos RTS

Em um jogo de estratégia em tempo real, ou RTS, cada jogador possui um número limitado de unidades controláveis individualmente, e tem como objetivo vencer ou controlar um ou mais oponentes usando as suas unidades e toda estratégia disponível possível. O termo "tempo real" indica que o jogo se desenrola continuamente e sem pausas, ao contrário dos jogos de estratégia em turnos, onde cada jogador possui um turno com tempo limitado para realizar as suas jogadas, e em seguida passa a sua vez para que o próximo jogador realize sua jogada.

As unidades controladas pelo jogador podem ser dos mais variados tipos, dependendo somente da temática do jogo. Podem ser tanto unidades de combate, como veículos, tanques, soldados, como também unidades construtoras, ou de apoio, ou qualquer outro tipo relevante para o tema do jogo. Dependendo do tipo de jogo, as construções, estruturas de apoio, e outros elementos de jogo também podem ser considerados como unidades, assim como nem toda unidade deve ser necessariamente controlável pelo jogador, podendo haver unidades totalmente autônomas ou não controláveis.

Os jogos RTS possuem uma ligação íntima com alguns dos conceitos de inteligência artificial, já que parte dos elementos de um jogo de RTS podem ser vistos, ou mapeados, como elementos de um sistema multiagentes, por exemplo.

1.3.3.1 Jogador

Em alguns jogos eletrônicos em geral, e nos jogos de RTS em particular, o termo "jogador" também pode fazer referência não só ao jogador humano, como também aos jogadores controlados pelo computador ou uma inteligência artificial.

Enquanto o jogador humano emite comandos de alto nível ou diretamente direcionados para as unidades do jogo, com um jogador controlado por computador os comandos para as unidades partem de um módulo de inteligência artificial especialmente construído para ler o estado da simulação do jogo e procurar vencer os adversários através das diversas possibilidades de gerenciamento das unidades disponíveis. As mesmas regras ou limites de controle de unidades para o jogador humano também costumam valer para o jogador controlado por computador, a fim de manter o jogo entre homem e computador o mais justo possível.

1.3.3.2 Time

A fim de facilitar a criação de jogos *multiplayer*, um ou mais jogadores podem ser agrupados em times, onde cada jogador dentro de um time joga de maneira cooperativa com seus companheiros, se existirem, e de maneira competitiva com os oponentes de outros times.

1.3.3.3 Recursos

O recurso, em um jogo típico de estratégia, equivale ao dinheiro ou material de construção disponível para que o jogador produza mais unidades ou outros elementos de jogo que ajudem o jogador a atingir o objetivo mais rápido ou com mais sucesso do que seus adversários. De uma maneira geral, os recursos devem ser coletados pelo jogador para que possam ser usados, porém uma nova tendência nos jogos RTS atuais é a geração automática dos recursos dependente da estratégia escolhida pelo jogador e das opções de jogo ativas no momento.

1.3.3.4 Unidades

As unidades são os principais objetos controlados pelo jogador em um jogo RTS. Basicamente, cada unidade possui uma lista limitada de ações que podem ser executadas ao comando do jogador, e cabe ao jogador informar à cada unidade o que deve ser feito, como deve ser feito, e onde deve ser feito. Uma unidade típica de um jogo de guerra pode ser um soldado, ou um tanque, contudo, nem todas as unidades devem poder se mover necessariamente, como no caso de fábricas, e também nem todas as unidades podem receber comandos, como por exemplo uma torre de defesa.

Os jogos RTS atuais estão procurando mudar o paradigma de gerenciamento de unidades com uma gradativa facilitação no envio de comandos do jogador para as unidades, através do envio de comandos para grupos de unidade, ou de comandos de alto nível que mudam o estado da unidade e permitem que a mesma atue de um modo mais autônomo em relação aos comandos do jogador. Outrossim, tipicamente as unidades são criadas dentro de estruturas fixas, como fábricas, centros de treinamento, e outros pontos de treinamento, ao comando do jogador, gastando uma quantidade de recursos e ocupando por um tempo a capacidade de produção da estrutura fabril.

Pela natureza da definição e das atribuições das unidades em um jogo RTS, as mesmas podem ser consideradas como sendo os agentes de uma simulação multiagentes representada pelo ambiente de jogo, e por conseguinte, os conhecimentos estabelecidos sobre sistemas multiagente podem ser usados ou adaptados para se melhorar ou complementar a qualidade da simulação do jogo, ou modificar o comportamento dos agentes.

1.3.3.5 Estruturas

Mesmo podendo ser tratadas de maneira diferente das unidades, as estruturas podem ser vistas como unidades que não se movem, ou que não recebem comandos da maneira convencional como as unidades. Um exemplo de uma estrutura pode ser uma fábrica, ou então um centro de comando. Geralmente as estruturas são construídas por unidades construtoras, ao comando do jogador, gastando uma quantidade de recursos, e tomando algum tempo útil da unidade construtora para que a construção seja completada.

Do ponto de vista de sistemas multiagente, dependendo da funcionalidade fornecida pela estrutura, a mesma pode ser considerada como um agente, que pode modificar o ambiente ou buscar seus próprios objetivos, ou ser considerada como um objeto passivo, que serve apenas para ser manipulado pelos agentes. A título de exemplo, uma fábrica ou uma torre de defesa poderiam ser considerados agentes, e uma ponte ou um muro poderiam ser considerados objetos.

1.3.3.6 Campo do jogo

O campo de jogo, também chamado de mapa, ou terreno, é a representação do espaço físico da simulação. Todas as unidades são posicionadas no campo, e a ação transcorre com as unidades pertencentes aos diversos jogadores recebendo comandos dos mesmos e executando suas ações respectivas dentro do campo de jogo, e possivelmente interagindo entre si.

Seguindo a análise da simulação do jogo da ótica de um SMA (Sistema multiagente), o campo de jogo representa o ambiente do SMA, onde se localizam todos os agentes e objetos, e onde transcorre a ação do sistema. Considerando as principais propriedades de um ambiente de um sistema multiagente, o campo de jogo pode ser classificado como:

- Completamente observável: os sensores dos agentes podem ter acesso à qualquer informação do mapa. Possíveis bloqueios ou limitação nos sensores podem ser empregados para atingir algum objetivo específico na simulação.
- Estocástico: o campo de jogo, em geral, não é determinístico, devido à complexidade de fatores envolvidos na simulação, incluindo jogadores humanos e possíveis conexões de rede em um jogo *multiplayer*.
- Dinâmico: o jogador e seus agentes controlados podem, e normalmente devem, modificar o ambiente, além de poderem trabalhar ao mesmo tempo, e em tarefas diferentes, o que torna a natureza do ambiente dinâmica.
- Discreto: devido à natureza numérica simbólica finita dos computadores, o tratamento dos estados possíveis do ambiente e dos sensores dos agentes é discreto, apesar da tentativa de fornecer um aspecto de continuidade dos fatores envolvidos através do uso de variáveis de programa que utilizam tipos de dados real, ou alguma variação de dados de ponto flutuante.
- Multiagente: em uma simulação normal de jogo, cada um dos jogadores possuirá no mínimo 1 unidade controlável, para garantir que haja uma disputa possível entre os jogadores.

1.3.3.7 Final de jogo

O final de jogo ocorre quando um time domina o time opositor, seja pela destruição de todas unidades do adversário, seja pela obtenção de algum objetivo estratégico, como por exemplo o acúmulo de uma quantidade determinada de recursos, ou o domínio de uma localização estratégica no mapa.

Habitualmente o final de jogo é regido por uma ou mais condições de finalização. As ações de alto nível selecionadas pelos jogadores, bem como os comandos ou decisões tomadas pelas unidades de forma autônoma, tendem a ser influenciadas pela condição de finalização de jogo, na medida em que determinadas decisões durante o jogo podem ser melhores do que outras, e os jogadores e seus agentes controlados tendem a tomar decisões visando maximizar seu ganho, ou mesmo diminuir o ganho do adversário. Assim, em um jogo em que o objetivo é a conquista de todos os pontos estratégicos do mapa, um jogador poderia focar em mobilidade e ações rápidas, e em um jogo em que o objetivo é a destruição de todos os oponentes, um jogador poderia focar em se defender, ou então em fortalecer suas unidades de ataque.

1.4 Estrutura do trabalho

No Capítulo 1 foram apresentadas definições para alguns conceitos importantes utilizados ao longo deste trabalho. Também é feito um paralelo entre alguns conceitos da área de informática e inteligência artificial, e conceitos presentes em jogos de estratégia.

No Capítulo 2 é feito um apanhado sobre os principais trabalhos relacionados à área de planejamento de caminhos em jogos de computador, também sendo mostrado um pouco mais do principal e mais usado algoritmo planejador de caminhos em jogos.

O Capítulo 3 mostra os detalhes técnicos do algoritmo planejador de caminhos proposto em (SIL 2009). No capítulo, é apresentada a forma de aplicação do método conforme proposto, bem como alguns problemas que podem resultar da aplicação do algoritmo.

O Capítulo 4 apresenta as informações referentes ao jogo de estratégia escolhido para uso nos testes a serem efetuados. Além das principais informações técnicas sobre o jogo, são mostrados detalhes à respeito do planejador de caminhos nativo já existente no jogo, e que servirá de comparativo de funcionamento no capítulo de testes. Por fim, são mostrados os detalhes da integração do planejador proposto por (SIL 2009) dentro do jogo escolhido.

Os resultados obtidos dos testes referentes à implementação do planejador de caminhos dentro do jogo são mostrados no Capítulo 5. São mostradas as configurações utilizadas para a realização dos testes, bem como gráficos e tabelas apresentando os resultados mais relevantes ou expressivos dos testes. Ao final do capítulo são mostradas algumas propostas de modificação ou otimização da implementação do algoritmo, bem como alguns resultados obtidos à partir das modificações.

Para finalizar, o Capítulo 6 apresenta as conclusões obtidas ao longo da realização deste trabalho. Juntamente com as considerações finais, são propostos alguns tópicos relacionados que podem ser utilizados como melhoria ou expansão deste trabalho, ou mesmo criação de trabalhos novos relacionados.

2 TRABALHOS RELACIONADOS

O planejamento de caminhos é um assunto extensivamente explorado no mundo dos jogos, onde o programador fica encarregado de configurar o tratamento de muitos personagens autônomos, e idealmente fornecer um comportamento realístico para os mesmos. Contudo, é muito difícil produzir movimentos realísticos usando técnicas de controle global dos personagens. Em contrapartida, levando em consideração características individuais de cada personagem pode ser uma tarefa computacionalmente pesada. Em vista disso, muitos dos métodos propostos na literatura de computação gráfica não levam em consideração o comportamento individual de cada agente, comprometendo o realismo do planejador de caminhos.

Kuffner (KUF 1998) propôs um método onde o terreno é mapeado em uma malha 2D, e o caminho é calculado usando um algoritmo de programação dinâmica como Dijkstra. Conforme Kuffner, a técnica é rápida o suficiente para ser usada em ambientes dinâmicos.

O desenvolvimento de algoritmos de planejamento aleatorizado de caminhos, especialmente os algoritmos de PRM (*Probabilistic Roadmaps*) (KAV 1996), permitiram a geração eficiente de caminhos em ambientes muito grandes e complexos. Existem muitos trabalhos seguindo este caminho (CHO 2003) (PET 2002), e na maioria dos métodos, o desafio está mais próximo da geração de movimentos realistas do que em encontrar um caminho válido.

Tecchia *et al.* (TEC 2001) propuseram um sistema de aceleração de desenvolvimento de comportamentos para agentes, através de um sistema de regras locais que controlam o comportamento dos agentes. As regras são governadas por quatro diferentes níveis de controle, cada um representando um aspecto diferente de comportamento do agente. Os resultados apresentados mostram que, para um modelo muito simplificado de comportamento, o desempenho do sistema pode atingir níveis interativos mínimos aceitáveis.

Campos potenciais também podem ser usados para gerar diretamente o movimento de um agente, podendo dispensar por completo a fase de planejamento de caminho (RUS 2004), o que pode ser um emprego interessante em um jogo, já que a eliminação de custo computacional de planejamento inicial pode ser relevante dependendo do tamanho da simulação e de outras características.

Em resumo, muitos dos métodos não levam em conta as características individuais de cada personagem, seus desejos ou necessidades, e por vezes podem gerar movimentos artificiais e pouco realistas. Alguns dos métodos apresentam modificações nos conceitos estabelecidos, de forma a melhorar o comportamento dos agentes, contudo a necessidade de se manter uma velocidade mínima da simulação, com resposta interativa, limita as ações para se obter sistemas em que o número de agentes é muito grande e o comportamento seja realista,

forçando os pesquisadores à buscar a linha de meio-termo entre realismo e velocidade.

Dos algoritmos de busca disponíveis, o A-Star (A*) é o mais utilizado em jogos computadorizados (PER 2005), tendo emergido como o mais comum algoritmo para planejamento de caminhos dentro da IA dos jogos (DEL 2008). A-Star pode ser resumido como uma busca em largura (*breadth first search*) em grafos (DEL 2008).

O algoritmo utiliza uma função heurística que determina a qualidade de cada um dos estados possíveis (nodos ou células), por meio de uma estimativa do custo da melhor rota até o destino, passando pelo nodo atual (PER 2005).

Pelo seu funcionamento, é considerado um algoritmo completo e ótimo: dada uma função heurística admissível, se o problema tiver uma solução, esta vai ser encontrada, e a solução encontrada será sempre a de menor custo (PER 2005). Por heurística admissível, se entende que seja uma função que nunca superestime o custo para alcançar o objetivo. Um exemplo de heurística admissível é a distância em linha reta. A distância em linha reta é admissível porque o caminho mais curto entre dois pontos quaisquer é uma linha reta, e assim a linha reta não pode ser uma superestimativa (RUS 2004).

Contudo, o caminho gerado pelo algoritmo A-Star por vezes pode parecer muito artificial, ou pouco realista. O fato do caminho encontrado ser baseado na otimização da distância e do custo ou função heurística das células pode fazer com que o caminho resultante, apesar de ótimo, contenha movimentos retos ou angulares visualmente artificiais.

A Figura 2.1 mostra um exemplo de caminho gerado pelo método A-Star, em que o caminho gerado (apresentando na figura pelos pontos vermelhos), apesar de ser ótimo conforme a definição exposta anteriormente, apresenta alguns segmentos pouco realistas e totalmente artificiais. Um agente caminhando sobre um mapa conforme o caminho em questão pareceria estar andando em movimentos ortogonais e diagonais, de uma maneira totalmente artificial.

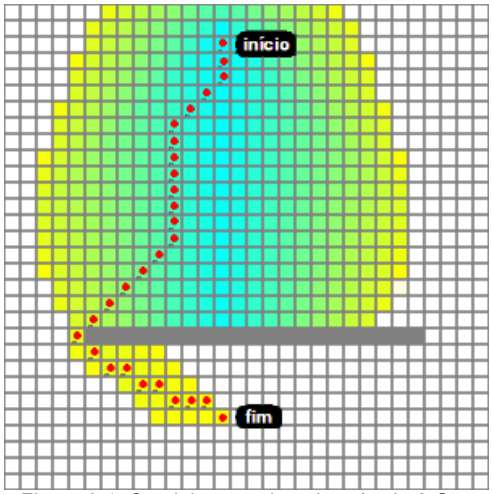


Figura 2.1: Caminho gerado pelo método A-Star

Fonte: http://modoogle.com/a-star/

3 O ALGORITMO PLANEJADOR DE CAMINHOS

O algoritmo planejador de caminhos implementado, proposto por Silveira *et al.* (SIL 2009), utiliza o conceito de campos potenciais, juntamente com funções harmônicas e resolução numérica de problemas de valor limite[4], para controlar o comportamento de direção e caminhamento de agentes em ambientes dinâmicos. Resultados teóricos e práticos da aplicação de algoritmos similares já foram mostrados em inúmeros outros trabalhos, incluindo: utilização do planejamento de caminhos usando campos potenciais em simulação de controle de robôs exploradores de ambiente (TRE 2006); testes de desempenho da aplicação do algoritmo utilizando computação em GPU (FIS 2008); além de animações e vídeos mostrando resultados práticos do caminhamento dos agentes em ambientes estáticos, dinâmicos, e de multiagentes (SIL 2010).

Pela natureza da técnica desenvolvida, o algoritmo apresenta uma relevância prática direta para aplicação na área de jogos RTS, que se caracterizam, no geral, pela existência de dezenas a centenas de agentes trabalhando de forma independente ou também cooperativa em um ambiente dinâmico.

3.1 Princípios fundamentais

Silveira *et al.* (SIL 2009) descrevem o algoritmo proposto como sendo um planejador de caminhos baseado em problemas de valor limite (BVP - *Boundary value problem*), e aproveitando a definição dada, este trabalho fará referência ao algoritmo proposto chamando-o de **planejador BVP**.

Conforme proposto, o planejador BVP se propõe a gerar caminhos realísticos usando estratégia de geração de fase única, onde o caminho é gerado dinamicamente enquanto o agente se movimenta, sendo que o caminho gerado já se encontra suavizado. A proposta também assume que os caminhos realistas gerados derivam das características pessoais humanas e do estado interno, dessa forma variando de pessoa para pessoa (SIL 2009).

O planejador BVP usa o conceito de campos potenciais com o objetivo de obter um vetor gradiente que represente a direção sugerida a ser seguida pelo agente. Contudo, Silveira *et al.* salientam que o uso clássico de campos potenciais resulta em mínimos locais[5] e caminhos não naturais, e para contornar o problema e gerar campos potenciais livres de mínimo local e que produzam caminhos realistas, é proposta a geração do campo potencial através da solução numérica de uma equação diferencial parcial com condições de contorno, ou seja, um

[4] *Boundary value problem* (BVP).

[5] Mínimos locais presentes no campo potencial podem levar o agente a ficar preso em determinadas áreas do mapa, sem possibilidade de atingir o objetivo.

problema de valor limite (BVP). As condições de contorno são vitais ao processo, indicando quais regiões do ambiente são obstáculos, e quais são objetivos, uma vez que a configuração de obstáculos influencia no caminho escolhido pelos agentes (SIL 2009).

3.2 Campo potencial

Para a geração do campo potencial sem mínimos locais, o planejador BVP usa o método de cálculo de funções harmônicas, onde o campo potencial é o resultado da solução da Equação de Laplace, tendo sido escolhida por não apresentar mínimos locais. Juntamente com a Equação de Laplace, é empregada uma condição de contorno no campo, onde o potencial dos contornos dos obstáculos tem o valor 1, e o potencial nas regiões de objetivo tem o valor 0. Em termos de BVP, ajustando-se o valor da função nos contornos do problema é chamado de Condição de Contorno de Dirichlet[6] (SIL 2009). Sendo assim, os caminhos são gerados usando a informação de potencial calculada através da solução numérica da seguinte equação:

$$\nabla^2 p(\boldsymbol{r}) + e\,\boldsymbol{v}.\nabla p(\boldsymbol{r}) = 0 \tag{3.1}$$

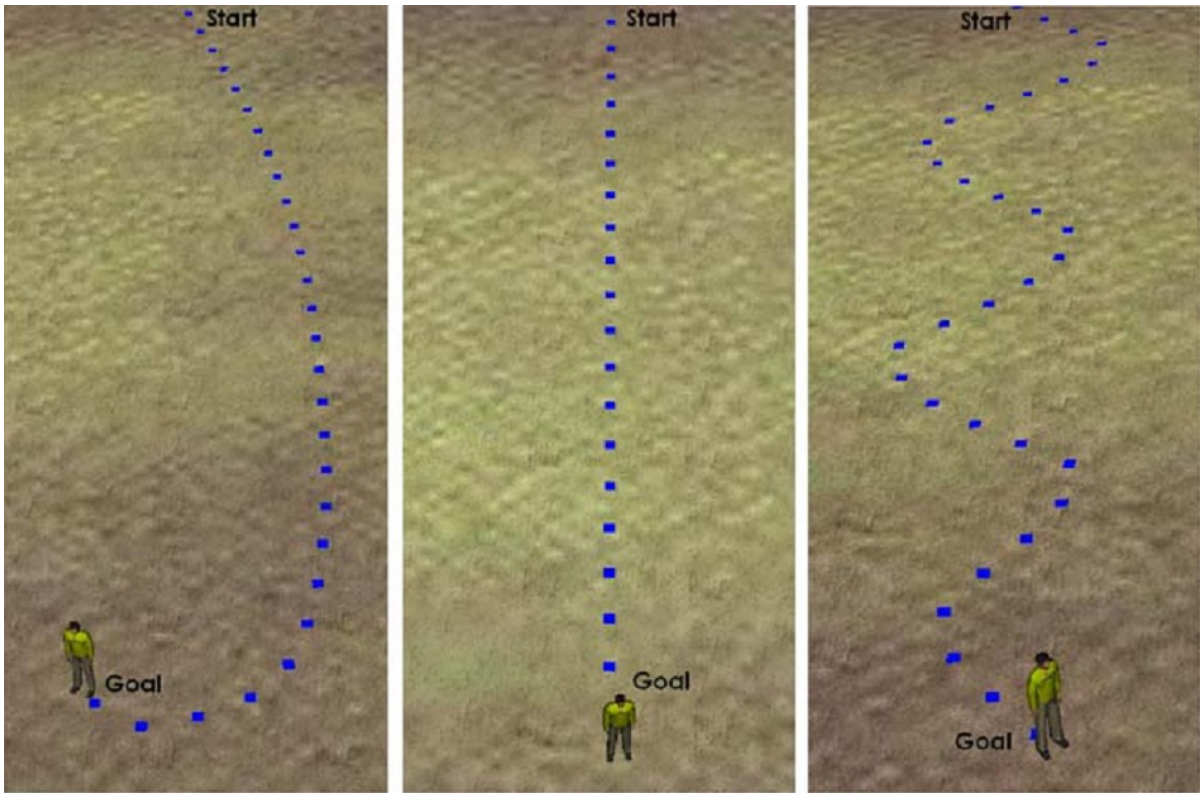
Na equação, **p(r)** é o potencial na posição $r \in \mathbb{R}^2$, **e** é um valor escalar, e **v** é um vetor unitário. Conforme Silveira *et al.*, através de um ajuste fino dos parâmetros **e** e **v**, podem ser obtidos comportamentos realísticos de movimentação dos agentes. O vetor **v**, chamado de vetor de comportamento, pode ser considerado como uma força externa que puxa o agente na sua direção, enquanto que o parâmetro **e** pode ser visto como a força de influência do vetor **v** no comportamento do agente. Quando o valor de **e** for igual a 0, a Equação 3.1 será reduzida para:

$$\nabla^2 p(\boldsymbol{r}) = 0 \tag{3.2}$$

que corresponde à Equação de Laplace.

Em Silveira *et al.* (SIL 2009) é salientado que, com o valor de **e** igual a 0, em um ambiente de interiores o agente tende a seguir um caminho equidistante das paredes, o que não se parece muito com o comportamento de humanos, que não caminham seguindo distâncias equidistantes da parede, e por isso propõe o uso dos parâmetros **e** e **v**. A Figura 3.1 mostra um exemplo de possibilidade de variação do caminho do agente através de mudanças nos valores do vetor de comportamento e influência.

[6] *Dirichlet boundary condition*.



(a) e *= -1.0 e* v *= (1, 0)* *(b)* e *= 0* *(c)* e *= -1.0 e* v *= (1, sin(0.6t))*

Figura 3.1: Diferentes caminhos seguidos pelo agente.

Fonte: (SIL 2009)

3.2.1 Cálculo do campo potencial

Seguindo a condição de contorno de Dirichlet, cada célula do campo potencial é inicializada com 0 ou com 1, conforme a mesma seja uma célula contendo o objetivo, ou um obstáculo, respectivamente. As células livres também são inicializadas com 1, porém têm seu valor posterior computado através do método de relaxamento de Gauss-Seidel, conforme a Equação 3.3:

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{e((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \tag{3.3}$$

onde, $p_c = p_{i,j}^{t+1}$, $p_b = p_{i,j+1}^{t}$, $p_t = p_{i,j-1}^{t+1}$, $p_r = p_{i+1,j}^{t}$, $p_l = p_{i-1,j}^{t+1}$, $v = (v_x, v_y)$. A Figura 3.2 mostra a representação das células.

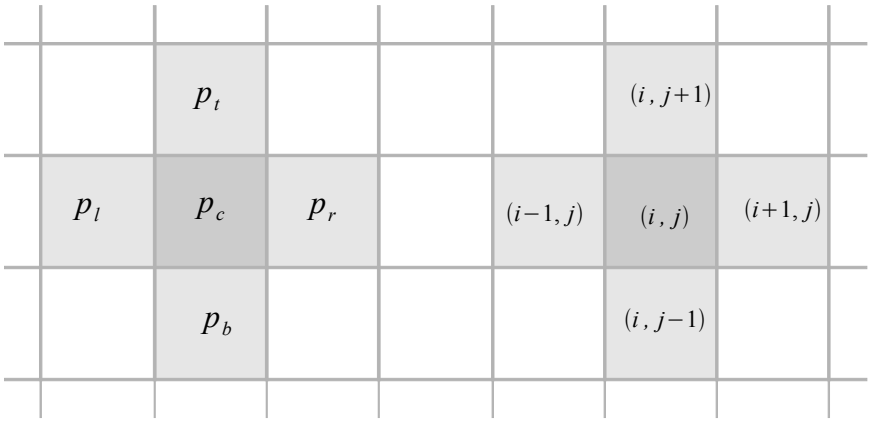


Figura 3.2: Representação dos pontos no grid.

Fonte: (SIL 2009)

Na Equação 3.3 o parâmetro **v** deve ser um vetor unitário, e o parâmetro **e** deve estar no intervalo (-2, 2), a fim de que não sejam gerados efeitos oscilatórios no caminho, o que poderia não garantir que o agente evite obstáculos ou atinja o objetivo.

3.3 Direção do agente

Depois do cálculo do campo potencial, o agente se movimenta seguindo a direção do gradiente descendente do potencial na sua posição (i, j), conforme mostrado na Equação 3.4:

$$(\nabla p)_{(i,j)} = \left(\frac{p_{i+j,j} - p_{i-1,j}}{2}, \frac{p_{i,j+1} - p_{i,j-1}}{2} \right) \tag{3.4}$$

A Figura 3.3 mostra um exemplo de campo potencial aplicado em um mapa. As flechas amarelas representam o vetor gradiente descendente, que aponta para o objetivo final, em verde.

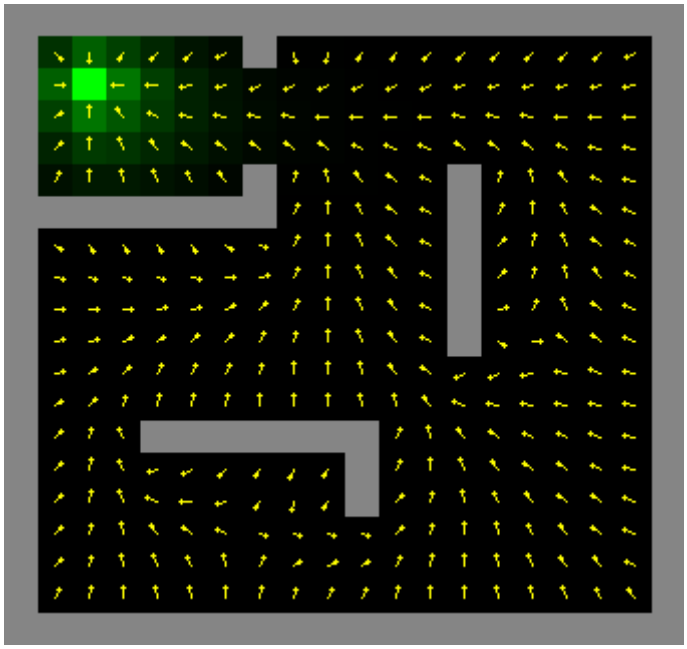


Figura 3.3: Campo potencial aplicado em um mapa.

O gradiente descendente já fornece uma maneira de controlar a direção do agente, porém Silveira *et al.* (SIL 2009) informam sobre problemas de reações abruptas do agente ao encontrar obstáculos dinâmicos muito próximos, pois os obstáculos produzem um campo repelente muito forte que resulta em mudanças bruscas na direção do agente.

Para contornar o problema, a posição atual do agente é ajustada por

$$\Delta d = v(\cos(\varphi^t), \sin(\varphi^t)) \tag{3.5}$$

onde, v é a máxima velocidade do agente, e φ^t é:

$$\varphi^t = \eta\, \varphi^{t-1} + (1-\eta)\, \zeta^t \tag{3.6}$$

onde, $\eta \in [0,1)$ e ζ é a orientação do gradiente descendente na posição atual do agente.

O valor η representa um fator de distribuição de força entre a direção anterior do agente, e a direção obtida do gradiente descendente, de forma que, quanto mais próximo de zero for η, maior será a influência do gradiente, até o ponto em que quando η = 0, a direção do agente será calculada apenas levando em conta a direção do gradiente. Com η = 0.5 a direção do gradiente e a direção anterior do agente serão combinadas com igual peso para fornecer a nova direção.

Mesmo com o cálculo ponderado da direção levando em conta a direção anterior do agente, em Silveira *et al.* ainda é salientado que, se o ambiente for desordenadamente cheio de obstáculos, o comportamento do agente ainda não será real, e colisões poderão acontecer. Para resolver esse problema, Silveira *et al.* mostram a inclusão de um controle de velocidade do agente, usando a seguinte equação:

$$\Delta \boldsymbol{d} = v(\cos(\varphi^t), \sin(\varphi^t))\, \psi(|\varphi^{t-1} - \zeta^t|) \tag{3.7}$$

onde a função $\psi : \mathbb{R} \rightarrow \mathbb{R}$ é:

$$\psi(x) = \begin{cases} 0, & se\ x > \pi/2, \\ \cos(x) & caso\ contrário. \end{cases} \tag{3.8}$$

Dessa forma, se a diferença entre a direção anterior, e a direção do gradiente descendente atual for maior do que $\pi/2$, então existe uma forte chance de colisão, e a função ψ retorna 0. Caso contrário, a velocidade do agente muda proporcionalmente ao risco de colisão, dado por cos(x).

3.4 Estratégia de utilização do método

A aplicação do planejamento de caminhos proposto por (SIL 2009) envolve a utilização de um mapa local para cada agente, e de um mapa global para cada objetivo.

3.4.1 Mapa global

O mapa global é formado por um campo potencial conforme descrito anteriormente, em que cada célula do campo potencial representa uma área quadrada do mapa de navegação do ambiente. Para delimitar a capacidade de navegação, o campo potencial do mapa global é rodeado por obstáculos estáticos.

3.4.2 Mapa local do agente

Cada agente possui um mapa local contendo as informações sobre o ambiente obtidas pelos seus sensores. Este mapa está centralizado na posição atual do agente, e representa uma fração diminuta do mapa global.

O mapa local é divido em 3 regiões, conforme mostra a Figura 3.4: zona de atualização (*u-zone*), zona livre (*f-zone*) e a zona de borda (*b-zone*). Cada célula do mapa representa uma célula real do ambiente original, e armazena um valor de potencial

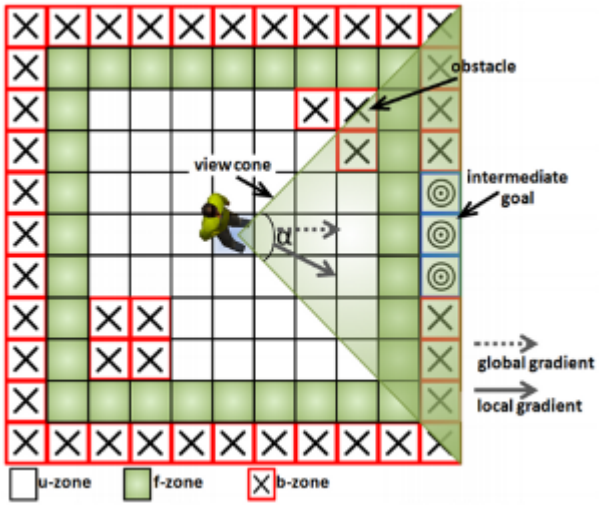


Figura 3.4: Mapa local do agente.

Fonte: (SIL 2009)

Para navegar em um ambiente, o agente usa seus sensores para atualizar seu mapa local com a informação sobre os obstáculos e outros agentes. O sensor do agente forma um cone de visualização, conforme mostrado na Figura 3.4. As células da *u-zone* dentro do cone de visão do agente que correspondem a obstáculos ou outros agentes recebem o valor de 1.

Usando o mapa global, para cada agente é calculado um vetor gradiente descendente da posição atual do agente no mapa global, e a direção obtida é usada para gerar o objetivo intermediário na borda do mapa local do agente. Assim, a célula ou células de *b-zone* que contêm o objetivo intermediário são marcadas como objetivo, e com valor de potencial igual a 0. As outras células da *b-zone* são marcadas como obstáculos, contendo o valor de potencial igual a 1.

As células da *f-zone* são sempre consideradas livres, mesmo que contenham obstáculos. O objetivo da *f-zone* é sempre garantir que exista uma propagação da informação sobre o objetivo para as células associadas à posição do agente.

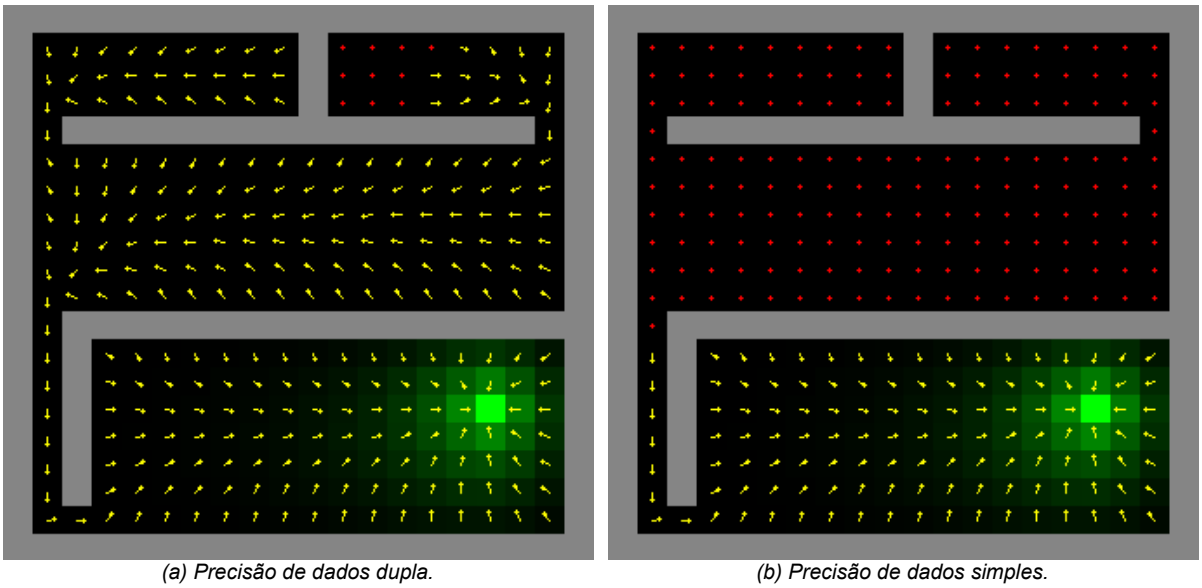
Após a geração do mapa local do agente, o deslocamento do passo atual do agente é calculado usando-se a Equação 3.8 e o gradiente descendente obtido do mapa local, e a posição atual e direção do agente são atualizados. Este processo é repetido para cada passo da simulação, para cada agente que esteja se movendo.

3.5 Planificação

Apesar da equação de cálculo do campo potencial ter sido escolhida de forma que não sejam gerados mínimos locais, durante o cálculo pode ocorrer um problema, chamado de

planificação (*flatness*), devido à precisão numérica limitada dos computadores (SIL 2009).

A Figura 3.5a mostra um exemplo da ocorrência do efeito de *flatness*. Os pontos vermelhos indicam células em que o vetor gradiente resultou em um vetor de tamanho zero, devido à proximidade do valor 1 (em alguns casos, as células são arredondadas para o valor 1) de valores de potencial das células nessa região. Na simulação da figura foram usados valores de ponto flutuante de precisão dupla para o relaxamento dos potenciais. A Figura 3.5b mostra a mesma simulação, usando-se valores de ponto flutuante de precisão simples, o que mostra que a diminuição da precisão resulta em um aumento do efeito de planificação.



(a) Precisão de dados dupla. *(b) Precisão de dados simples.*

Figura 3.5: Problema da planificação e a precisão dos dados.

A ocorrência do efeito de planificação é indesejada, já que o gradiente computado do campo potencial das regiões em que ocorre o problema será nulo, impedindo cada agente situado na região de se mover (SIL 2009).

Em Silveira *et al.* é mostrado que o problema ocorre em regiões separadas do objetivo por passagens estreitas ou distantes muitas células do objetivo, e proposto como saída para o problema a criação de subobjetivos quando seja detectado que o agente se encontre em uma região de planificação.

4 O JOGO RTS

O jogo de estratégia em tempo real a ser usado teve como objetivo fornecer o ambiente simulado empregado nos testes realizados neste trabalho. Neste capítulo são mostradas algumas características e propriedades das principais *engines* de jogos de código aberto disponíveis para uso, bem como os motivos da escolha da *engine* selecionada. São mostradas também as informações gerais e pormenores técnicos do planejamento de caminhos do jogo escolhido, além de todos os detalhes referentes à integração do novo planejador de caminhos BVP dentro da *engine*.

4.1 Escolha da *engine* de jogos

Levando-se em consideração a natureza de controle multiagente da técnica do planejador de caminhos proposta por Silveira *et al.* (SIL 2009), os melhores jogos candidatos à receberem a implementação do planejador são justamente os jogos de estratégia em tempo real, já que, em sua maioria, apresentam como principal característica a simulação de um ambiente com múltiplos personagens interagindo em um mapa compartilhado e relativamente plano, e dessa forma, a busca e escolha de uma *engine de jogos* foi direcionada para as *engines* de jogos de estratégia.

4.1.1 Principais jogos de estratégia de código aberto

Várias *engines* de jogos de código aberto foram analisadas em busca de características que permitissem uma fácil integração de um planejador de caminhos, bem como permitissem a execução de testes a fim de verificar os resultados obtidos da integração do planejador. Dentre todas as *engines* analisadas, as mais relevantes encontradas para a execução deste trabalho foram:

- *Unity 3D* (UNI 2010): É uma ferramenta de autoria de jogos. Apesar de proprietária, possui uma versão gratuita, que pode ser usada tanto para fins educacionais quanto para fins comerciais (com algumas restrições). A *Unity* tem como enfoque a criação do jogo através da sua interface e ambiente de desenvolvimento, instigando o programador a usar os recursos avançados disponibilizados pela própria ferramenta.
    - Ambiente de desenvolvimento integrado.
    - Múltiplas plataformas: Windows, Mac, Web, Nintendo Wii, iPhone/iPad, Android, Xbox360, PlayStation 3.
    - Suporte à física via PhysX.

  - Escrita do jogo via projeto Mono, que é a implementação *open source* do *.NET Framework.*
  - Linguagens suportadas: JavaScript, C# ou Boo.

- *Unreal Engine* (UNR 2010): É uma *engine* de jogos escrita em C++, primeiramente usada no desenvolvimento do jogo *Unreal* em 1998, porém também usada como base para muitos outros jogos de sucesso, como *Gears of War, BioShock, Batman: Arkham Asylum*, e muitos outros.
  - Desenvolvida para uso em um jogo de tiro em primeira pessoa (FPS), porém também usada com sucesso em outros gêneros, como MMORPG e RPG
  - Linguagem C++, com suporte a *UnrealScript*
  - Licença proprietária, liberado para uso não-comercial
  - Múltiplas plataformas: Windows, Linux, Mac, Dreamcast, Xbox, Xbox 360, PlayStation 2 e PlayStation 3
- *Spring RTS* Engine (SPR 2010): É uma *engine* de jogos especialmente desenvolvida para trabalhar com jogos do tipo RTS. Através de uma arquitetura modular, a *Spring* foi construída de forma a fornecer todos os principais recursos de jogabilidade necessários para a construção de um jogo de estratégia, como por exemplo: gerenciamento de unidades, estruturas e recursos; manipulação dos detalhes do sistema de armas e batalha; simulação de física e detecção de colisão; entre outros. Dessa forma, a *Spring* não é definida como um único jogo de estratégia, mas sim como uma plataforma de execução de jogos completa, que permite a rápida criação de variados e ricos jogos de estratégia, minimizando o esforço de programação necessário.
  - Desenvolvida para uso como plataforma de jogos de RTS.
  - Linguagem C++, com suporte para *scripts* LUA.
  - Licença GPLv2 para a *engine.*
  - Arquitetura modular, para fácil configuração de inteligência artificial, tipo de jogo, e mapa.
  - Muitos pacotes de jogos prontos e mapas disponíveis.
  - Múltiplas plataformas: Windows, Linux.
  - Várias *Skirmish AIs* (ou *Bots*) já prontas e configuradas para controlar um time no lugar de um jogador humano, possibilitando partidas *single player*, apesar do enfoque para jogos *multiplayer.*
  - Planejador de caminhos nativo.
- *Irrlicht Engine* (IRR 2010): É uma *engine* de jogos 3D, de código aberto e escrita em C++. É muito conhecida pelo seu tamanho pequeno e boa compatibilidade com hardware novo e também antigo. Fornece o *framework* básico para a construção de um jogo ou aplicação multimídia em 3D.
  - Múltiplas plataformas: Windows, Linux, Mac, Windows CE.

- *Framework* básico para programação do jogo.
- Suporte a vários tipos conhecidos de arquivos multimídia.

4.1.2 A escolha da *engine* de jogos

A *engine* escolhida recebeu modificações no código fonte, de forma a acomodar o planejador de caminhos a ser integrado. Dessa forma, é de vital importância que a *engine* possuísse código aberto.

Não obstante, é interessante que sejam apresentadas diversas configurações diferentes de situações para o planejador de caminhos, a fim de verificar o comportamento do mesmo em situações diferentes.

Além disso, para a realização dos testes, várias partidas teriam de ser jogadas, e os dados de desempenho ou demais parâmetros da simulação seriam anotados para posterior conferência e verificação dos resultados. É interessante também que os dados obtidos pelo planejador de caminhos testado sejam comparados com os dados obtidos da execução de outro planejador de caminhos.

Todas as *engines* de jogo enumeradas fornecem alguns destes recursos, em menor ou maior grau.

A *Unity*, enquanto permite um rápido desenvolvimento do jogo através de seu ambiente de programação, possui o enfoque na criação de jogos à partir do zero. Para que fossem apresentadas as diversas configuração para teste do planejador, precisariam ser testados diversos jogos diferentes rodando na *Unity*, e os jogos precisariam ser de código aberto para que fossem modificados e integrados com o planejador. Além disso, a *Unity* não parece possuir um planejador de caminhos integrado ou que possa ser modificado, deixando a cargo do programador a criação de um.

A *Unreal Engine* possui o código totalmente disponível para modificação, permitindo uma fácil inclusão de um planejador de caminhos. Contudo, também parece não possuir um planejador de caminhos integrado ou que permita uma fácil modificação. Quanto ao teste em diversos jogos ou diversos ambientes de jogos, como a *Unreal Engine* serve apenas como base de programação para criação de um jogo maior, necessitaria que fossem criados jogos de teste ou sintéticos, ou então estudados e modificados diversos jogos prontos já existentes escritos para *Unreal Engine* e de código aberto, o que seria uma tarefa trabalhosa. Como o enfoque da *Unreal Engine* é FPS, também seria difícil encontrar jogos de RTS para teste.

A *Irrlicht Engine*, como a *Unreal Engine*, possui o código aberto e totalmente modificável. Contudo, a mesma não possui um planejador de caminhos nativo que possa ser modificado ou usado para controle das unidades. Além disso, poucos jogos de estratégia foram encontrados escritos para uso na *Irrlicht Engine*, e nem todos com o código aberto ou disponível para modificação, o que dificultaria o teste do planejador em ambientes diferentes.

A *Spring Engine* foi a candidata que se mostrou mais favorável para servir de base para a execução deste trabalho, pois:

- possui o código aberto, escrito em linguagem C++, o que facilita a execução das mudanças;
- foi construída e focada para uso em jogos de estratégia, e já fornece os principais recursos presentes em um jogo de estratégia;

- possui um planejador nativo que pode ser usado para fins de comparação;
- possui diversos tipos de mapas e jogos que podem ser configurados, permitindo uma variação do ambiente disponível para o planejador;
- possui um módulo de IA que permite que dois oponentes joguem sem intervenção do usuário, possibilitando uma automatização dos testes, e menor interferência humana nos resultados.

Sendo assim, apesar de seu tamanho relativamente grande, e complexidade do projeto, a *engine* escolhida foi a *Spring Engine* (SPR 2010), devido principalmente ao seu código aberto, sob licença GPL, e à sua característica de alta modularidade e possibilidade de configuração, se mostrando especialmente interessante quanto à implementação do planejador de caminhos, já que o seu sistema modular de integração de componentes de jogo se apresenta como uma ótima possibilidade de múltipla variação de cenários de teste para o planejador de caminhos. A versão da *Spring* usada neste trabalho foi a versão 0.81.2.

4.2 Informações gerais

A *Spring* é uma *engine* de jogos especialmente desenvolvida para trabalhar com jogos do tipo RTS, através da disponibilização de uma arquitetura modular, que fornece todos os principais recursos necessários para a construção de um jogo de estratégia. A Figura 4.1 mostra um exemplo de uma batalha entre robôs na *Spring*.

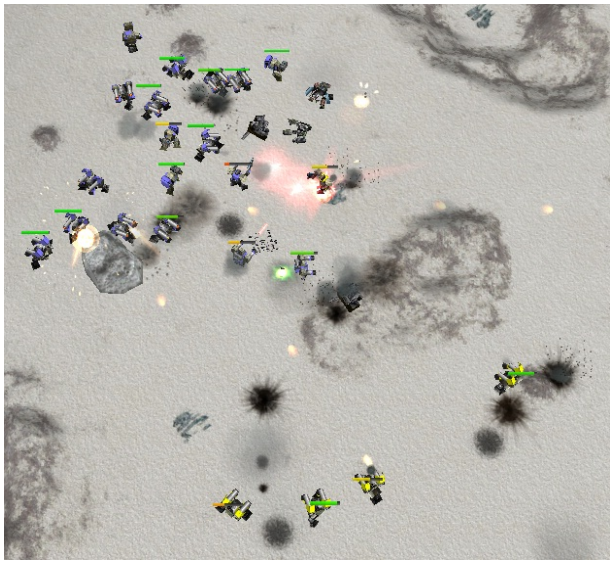


Figura 4.1: Batalha entre robôs na Spring.

Além da arquitetura modular, o código fonte da *Spring* foi escrito na linguagem C++, utilizando-se orientação a objetos e múltiplas bibliotecas de suporte comuns bem conhecidas, com o objetivo de obter uma aplicação altamente portável e estável rodando nas plataformas de sistema operacional mais comuns, como Windows e Linux.

Cada jogo na *Spring* é iniciado através da chamada ao arquivo executável da *Spring* passando-se o nome de um arquivo de texto, apelidado de *script* inicial, contendo a configuração da sessão de jogo. Tipicamente, o *script* inicial é criado por uma aplicação

externa[7], que encadeia o processo de jogo e abstrai do usuário os detalhes de uso da *Spring*. O *script* inicial contém as principais definições que podem variar à cada sessão de jogo, como o nome do arquivo do mapa, o nome do arquivo *Mod*, a configuração de times e jogadores, incluindo a definição dos componentes controlados pela IA e oponentes humanos.

O arquivo do mapa contém as definições visuais e também paramétricas do terreno de jogo, como por exemplo a elevação do terreno, obstáculos fixos, tipo de solo, e outras. Visando a facilidade de distribuição, o arquivo de mapa é autocontido, e possui todos os dados e informações necessárias para sua utilização, incluindo texturas, imagens, matrizes, malhas, e qualquer outro dado necessário para a renderização ou utilização do mapa na *Spring*.

O arquivo *Mod* contém as definições referentes aos tipos de unidades e características gerais do tipo de jogo, como por exemplo: informações sobre as unidades e estruturas, dados de física, movimentação e combate, características de equipamentos, malhas 3D dos objetos, texturas, imagens, e animações. O arquivo *Mod* também é autocontido, de forma a organizar o ambiente de jogo e permitir uma fácil distribuição de pacotes de um jogo contendo o arquivo *Mod* e diversos mapas. Os mapas não são vinculados ao *Mod* de jogo, de forma que mapas feitos para um tipo de jogo também podem ser carregados e usados em outros tipos de jogos.

A inteligência artificial na *Spring*, apelidada tão somente de IA pela *engine*, também é definida através de módulos externos, selecionados para a sessão de jogo via *script* inicial. Conforme a documentação fornecida no site (SPR 2010), a *Spring* fornece uma interface de conexão de inteligência artificial através de bibliotecas DLL, programas JAVA, e *scripts* LUA. Assim como os mapas, as IAs também não são vinculadas ao tipo de *Mod* de jogo, e nem aos mapas selecionados, podendo ser livremente selecionadas no *script* inicial, apesar de existirem algumas IAs especializadas em determinados tipos de mapa ou *Mod* de jogo. De qualquer maneira, algoritmos de IA mais genéricos precisam ler dinamicamente, através da interface fornecida pela *Spring*, a informação de terreno e *Mod* de jogo, a fim de tirar vantagens de diferentes tipos de configurações da sessão de jogo.

4.3 Posicionamento de objetos

A *Spring* considera estruturas como unidades que não podem se mover, e dessa forma podendo tratar unidades e estruturas genericamente e da mesma maneira.

Todas as unidades em jogo possuem a sua localização definida por um ponto 3D no espaço, contendo as coordenadas (x, y, z) em valores do tipo ponto flutuante. A janela de visualização do sistema gráfico, acessível através da API OpenGL, está configurada para coincidir com o sistema de coordenadas de posicionamento das unidades, e dessa forma nenhuma tradução de posicionamento é necessária entre o posicionamento das unidades e sua representação gráfica.

As coordenadas (x, y, z) das unidade estão mapeadas para utilizar o eixo y como sendo a altura do terreno, logo a posição da unidade no terreno é definida pelo par (x, z), e a altura do terreno é definida pelo ponto y. Unidades aquáticas ou voadoras podem conter o valor y diferente da altura do terreno. A carga do arquivo de mapa e alocação das unidades foi calculada de forma a mapear os valores de x e z dos objetos somente na faixa de valores positivos. Quanto ao ponto y, que representa a altura do terreno, geralmente é utilizado com

[7] Os jogadores costumam chamar a aplicação externa de *Looby*.

valor 0 indicando nível do mar[8], valor negativo indicando posição subaquática, e valor positivo indicando fora da água, terreno elevado, ou aéreo.

Além disso, alguns módulos e rotinas da *Spring* utilizam o plano referente ao posicionamento da unidade no mapa discretizado em quadrados, chamados de *square* pela *engine*, a fim de organizar o posicionamento dos objetos ao estilo de um tabuleiro de jogo com coordenadas discretas. Cada quadrado do tabuleiro possui o tamanho definido por uma constante com valor preestabelecido em 8 unidades. Assim, todos os quadrados possuem uma largura de 8 e comprimento de 8, e o mapeamento de valores de posicionamento real para posicionamento em tabuleiro devem levar em conta essa diferença, conforme a relação a seguir:

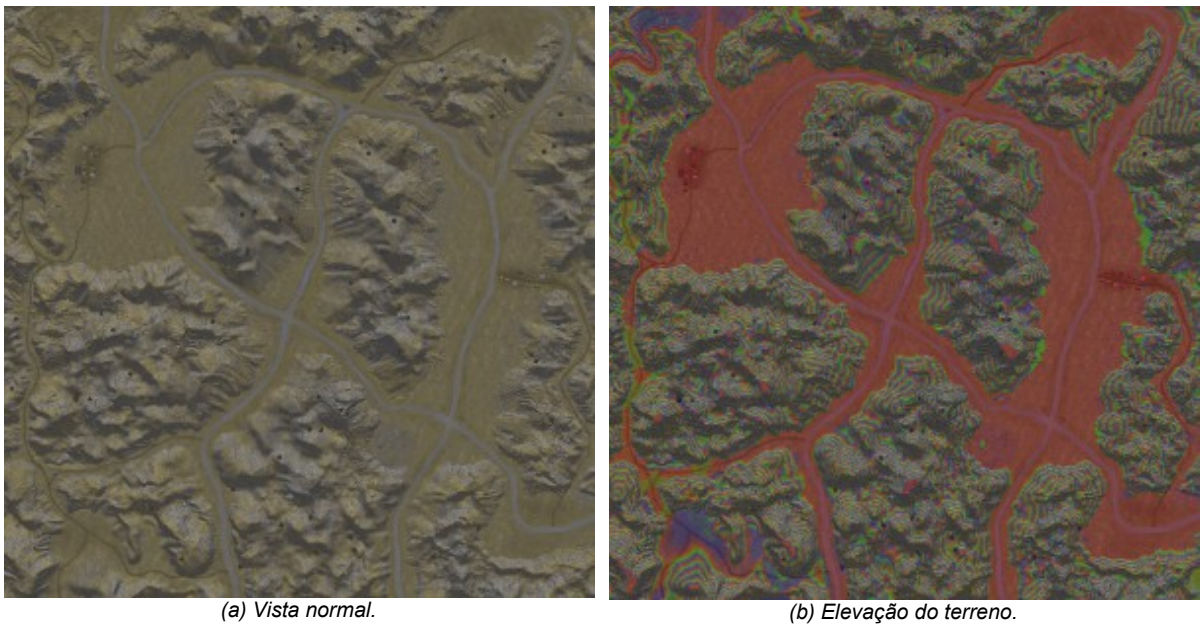
$$x_{tabuleiro} = \frac{x_{real}}{8}$$
$$y_{tabuleiro} = \frac{z_{real}}{8} \qquad (4.1)$$

4.4 Mapa de jogo

A simulação do jogo na *Spring* se passa como se estivesse ocorrendo sobre a superfície de um tabuleiro, a fim de simplificar os cálculos e tratamentos do jogo. Contudo, o terreno não está limitado à um plano 2D, podendo possuir montanhas, rios, penhascos, desfiladeiros e outras deformações topológicas, conforme a necessidade do desenvolvedor do mapa. A ideia é que as unidades devam se adaptar ao terreno e à sua topologia, e para isso o módulo de física também leva em conta a altura do terreno para efetuar os cálculos de velocidade da unidade conforme a inclinação do terreno, possibilidade de passagem, trajetória de projéteis, e até mesmo a deformação do terreno causada por explosões muito fortes.

A Figura 4.2a mostra um exemplo de vista aérea de um mapa composto por várias montanhas, e também algumas estradas em terreno baixo e plano. A Figura 4.2b mostra a variação de elevação do mapa, ao estilo de um mapa topográfico, onde as variações de cor mostram variação na elevação do terreno.

[8] Nível da água, na simulação do jogo.



(a) Vista normal. (b) Elevação do terreno.

Figura 4.2: Vista aérea de uma mapa.

4.5 Planejador de caminhos nativo

O planejador de caminhos nativo da *Spring* foi construído de forma a receber requisições de caminho entre dois pontos no mapa, vindas da *interface* do módulo de IA, ou então do módulo responsável pelo movimento terrestre. As requisições de caminho contêm informações sobre a unidade requisitante do caminho, dados de movimentação, o ponto de partida, o ponto de chegada, e o raio de chegada[9]. As requisições retornam um número identificador único, que a unidade deve utilizar posteriormente para solicitar detalhes sobre o caminho gerado.

Os dados de movimentação das unidades contêm todos os detalhes relevantes à movimentação da unidade, incluindo o tamanho, peso, inércia, velocidade, e tipo de terreno. A informação é basicamente utilizada para verificar o custo do terreno e possibilidade de passagem dos pontos intermediários para o caminho gerado.

De posse do identificador único de caminho, a cada passo da simulação do jogo (*frame* do jogo), a unidade pode solicitar que seja gerado um ponto intermediário dentro do caminho requisitado, que será usado como próximo subobjetivo para que a unidade possa chegar ao ponto final de destino.

O principal custo do terreno é chamado de modificador de velocidade (*speed mod)*, e tem como objetivo fornecer um número real entre 0 e 1 para cada célula do tabuleiro, indicando assim um fator multiplicador da velocidade da unidade passando pela célula. Valores de *speed mod* em 0 ou próximos de 0 indicam um terreno não passável ou de difícil acesso, como por exemplo uma rampa muito inclinada na borda de uma montanha, ou até mesmo um penhasco. Valores de *speed mod* em 1 ou próximos de 1 indicam terreno passável ou de fácil passagem, afetando de maneira tênue a velocidade atual da unidade. O valor de *speed mod* é calculado à

[9] Distância mínima do objetivo para considerar que a unidade atingiu o objetivo.

partir dos dados geográficos do terreno, como inclinação e tipo de solo, e também varia de acordo com as características de movimentação da unidade, como peso, direção e inclinação máxima possível. Sendo assim, o valor de *speed mod* para cada célula não pode ser calculado de antemão no início da simulação, já que os valores podem variar no decorrer da simulação, e também serão diferentes para cada tipo de unidade.

O planejador nativo utiliza um sistema de busca em largura no grafo representado pelo tabuleiro do mapa, usando uma variação do método usado no planejador de caminhos A*. Durante a geração do caminho, o planejador testa a possibilidade de passagem e custo dos quadrados do tabuleiro em relação aos seus 8 quadrados vizinhos diretos e diagonais[10]. O custo dos quadrados leva em conta a distância a ser percorrida, bem como o custo de travessia do quadrado em questão, considerando inclinação e dificuldade do terreno para passagem da unidade, utilizando os dados de movimentação para o cálculo da facilidade ou dificuldade. O custo dos quadrados também é modificado por um valor de "mapa de calor" (*Heat Map)*, gerado pela movimentação de cada unidade, de forma a fazer com que as unidades evitem seguir caminhos iguais, e assim melhorar a qualidade da animação gerada. O caminho gerado pelo planejador nativo é representado por uma lista de pontos a serem seguidos pela unidade, a fim de atingir o ponto de chegada utilizando-se um caminho de custo relativamente baixo, e que evite a passagem por obstáculos.

4.6 Integração do novo planejador de caminhos

4.6.1 Acesso ao planejamento de caminhos

O planejamento de caminhos é acessado pelos diversos módulos da *Spring* de uma maneira centralizada através de chamadas aos métodos de uma instância da classe principal de planejamento de caminhos, que fornece as principais chamadas ao planejador, incluindo a requisição de caminhos, requisição de pontos intermediários, atualização dos caminhos, e liberação de requisições. Toda e qualquer chamada referente ao planejamento de caminhos nativo passa obrigatoriamente por essa interface de acesso centralizada.

Os principais métodos relevantes existentes na interface do planejador de caminhos são:

- Requisição de caminho (*RequestPath*, em PathManager.cpp): a unidade informa o ponto de origem e o ponto do objetivo, recebendo um identificador numérico único do caminho requisitado.
- Liberação de caminho (*DeletePath*, em PathManager.cpp): a unidade informa que o caminho não é mais necessário, podendo ser liberado da memória.
- Requisição de próximo ponto (*NextWaypoint*, em PathManager.cpp): a unidade informa o identificador de caminho e a sua posição atual, e requisita o próximo ponto de subobjetivo que a unidade deve se direcionar. O ponto retornado é bem próximo, distante somente a alguns passos da simulação considerando a velocidade da unidade, de forma que a unidade faça o caminho em linha reta e não precise ser feito planejamento de caminho extra entre o ponto atual e o ponto retornado.
- Atualização de dados (*Update*, em PathManager.cpp): a rotina é chamada a cada

[10] Seguindo a ideia do algoritmo A*, nem todos os quadrados são testados, de forma a diminuir o custo computacional da busca..

passo da simulação, de forma a permitir que sejam atualizadas as estruturas internas do planejador.

4.6.2 Mudança do planejador de caminhos nativo

Devido à centralização das chamadas ao planejamento de caminho e ao baixo acoplamento de dados nas chamadas, sem a passagem de estruturas complexas do planejamento nativo, a retirada do planejador nativo e colocação do planejador BVP pode ser feita através do redirecionamento das chamadas aos métodos da classe de planejamento de caminhos, conforme segue:

- Requisição de caminho (*RequestPath*, em BVPPathManager.cpp): é criada uma estrutura de planejamento de caminhos BVP vinculada à unidade requisitante e ao ponto final de objetivo, sendo retornado o identificador numérico referente à requisição. Nesse ponto poderá ser criado o mapa global do objetivo e o mapa local do agente.
- Liberação de caminho (*DeletePath*, em BVPPathManager.cpp): a solicitação informa que o caminho não é mais necessário, e a estrutura do planejador BVP pode ser liberada da memória.
- Requisição de próximo ponto (*NextWaypoint*, em BVPPathManager.cpp): o planejador de caminhos BVP trabalha baseado em direção, e não por pontos, logo a maneira de simular esse funcionamento é retornar um ponto próximo do ponto atual da unidade na direção fornecida pelo planejador BVP no passo atual da simulação. A proximidade do ponto garantirá que não precise ser feito planejamento de caminho extra entre o ponto atual e o ponto retornado.
- Atualização de dados (*Update*, em BVPPathManager.cpp): para cada estrutura de planejamento BVP existente, será feita a leitura do mapa local do agente e atualização do campo potencial pelo processo de relaxamento.
- Requisição de próxima direção (*NextWaypointDir*, em BVPPathManager.cpp): a chamada é nova e necessária, sendo usada pela módulo de controle de movimento das unidades terrestres.

4.6.3 Controle de movimento das unidades

O módulo original de controle de movimento das unidades terrestres[11] (arquivo GroundMoveType.cpp) executa o seguinte algoritmo:

[11] Na *Spring* as unidades aéreas possuem total liberdade de movimento, e não utilizam controle de movimento, seguindo sempre em linha reta para seus objetivos. As unidades aquáticas usam o movimento terrestre.

para cada unidade com objetivo:
  se chegou no objetivo, encerra o movimento
  se chegou no subobjetivo,
    subobjetivo ← *requisita próximo ponto*
  fim se
  $direção = posição_{atual} - subobjetivo$
  acelera na nova direção
  $atualiza\ posição_{atual}$
  trata colisões e atualiza posição se necessário
fim para

Para acomodar as características do planejador de caminhos BVP, o algoritmo de controle de movimento foi modificado da seguinte forma (em GroundMoveType.cpp):

para cada unidade com objetivo:
  se chegou no objetivo, encerra o movimento
  $direção_{gradiente} \leftarrow$ *requisita próxima direção*
  $velocidade_{atual} \leftarrow calcula\ à\ partir\ de\ direção_{gradiente}\ e\ direção_{anterior}$
  $direção_{atual} \leftarrow calcula\ à\ partir\ de\ direção_{gradiente}\ e\ direção_{anterior}$
  $movimenta\ com\ velocidade_{atual}\ e\ na\ direção_{atual}$
  $direção_{anterior} \leftarrow direção_{atual}$
fim para

O controle de direção modificado agora requisita uma direção ao planejador de caminhos, ao invés de requisitar um próximo ponto, como era feito originalmente. A direção e velocidade real a ser seguida é calculada conforme proposto por Silveira *et al.* (SIL 2009). O teste de colisão padrão da *Spring* foi removido, a fim de garantir que o método proposto por Silveira *et al.* (SIL 2009) controle todos os aspectos do planejamento e da evasão de obstáculos.

4.6.4 Estrutura de planejamento BVP

Cada caminho requisitado ao planejador de caminhos gera uma estrutura de caminho BVP, que é referenciada por um identificador numérico único. A estrutura de caminho BVP é responsável por gerenciar a criação do mapa local da unidade, bem como fornecer uma atualização periódica do campo potencial do mapa local, e controlar a geração dos vetores gradientes descendentes referentes à posição da unidade dentro do mapa local.

O vetor gradiente do mapa global, que é usado para mapear o objetivo intermediário dentro do mapa local do agente, é gerado usando-se o planejador de caminhos nativo da *Spring*. O vetor de direção global é calculado à partir da diferença entre a posição atual do agente e o próximo ponto fornecido pelo planejador nativo, e então o objetivo intermediário é projetado nas bordas do mapa local seguindo a direção informada.

As informações de obstáculo e célula livre usadas para preenchimento do mapa local são construídas à cada passo da simulação à partir do módulo de informação sobre o terreno fornecido pela *Spring*. O módulo de informação sobre o terreno centraliza as consultas aos módulos referentes ao gerenciamento de movimentação, física, terreno e mapa, visando a obtenção da informação de possibilidade de passagem de uma determinada unidade relativa a cada quadrado do tabuleiro representado pelo mapa de jogo. São levados em conta diversos

aspectos da possibilidade de passagem em terreno, incluindo informações do tipo se um objeto bloqueia a passagem de outro, considerando-se a massa e capacidade estrutural de impacto, de forma a permitir que objetos pesados destruam objetos menores ao passarem por cima deles.

Além disso, não só as células contendo objetos são consideradas como ocupadas, assim como as células adjacentes, dependendo do tamanho da unidade envolvida. Como a *Spring* permite que as unidades possam ocupar um tamanho maior do que 1x1 célula, a construção da informação de bloqueio do mapa local leva em conta o tamanho da unidade, e dessa forma o mapa local é montado considerando-se os pontos em que o centro da unidade atual pode passar sem gerar uma intersecção entre os volumes das geometrias 3D da unidade e dos obstáculos.

A Figura 4.3 mostra um exemplo de informação de bloqueio e passagem do mapa local do agente projetada sobre a simulação do jogo. Os pontos amarelos indicam uma célula passável, e os pontos vermelhos indicam uma célula bloqueada. Também pode ser vista uma célula verde projetada na borda do mapa local, indicando o objetivo intermediário da unidade, bem como também podem ser vistas as bordas formadas pela *b-zone* (região vermelha ao redor do mapa) e *f-zone*[12] (borda livre junto à *b-zone*, na cor laranja) do mapa local.

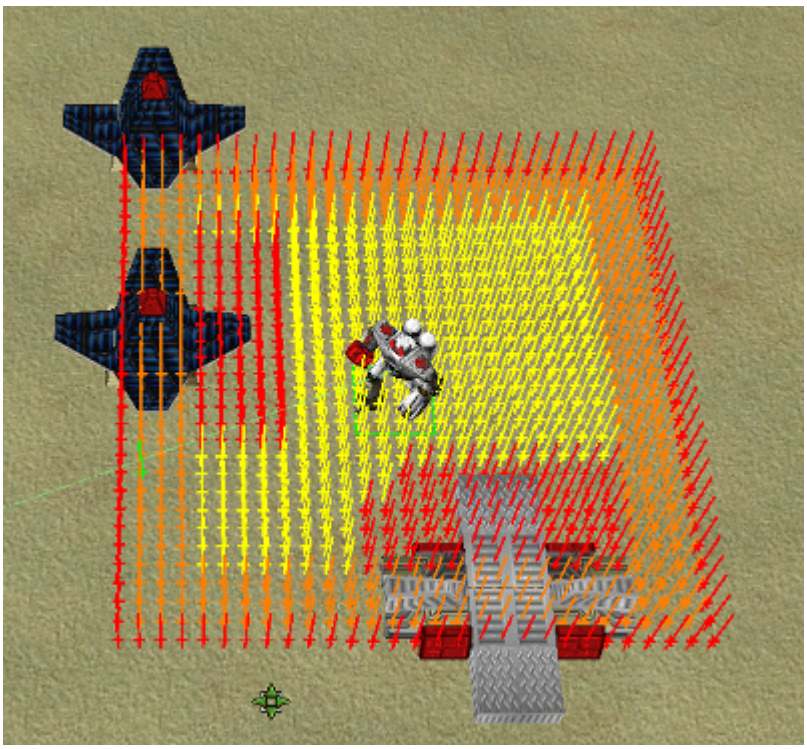


Figura 4.3: Informação de passagem e bloqueio do mapa.

O fator de velocidade *speed mod* também é usado para determinar os obstáculos baseados nas deformidades dos terrenos. Um valor experimental de corte de *speed mod* é usado para limitar a passagem em pontos muito inclinados ou íngremes, como paredes, penhascos e muralhas. Valores de *speed mod* maiores do que o valor de corte indicam que a velocidade possível na célula é alta[13], informando que a unidade pode atravessá-la, e valores de *speed mod* menores do que o valor de corte indicam que a célula é um lugar de difícil passagem ou

[12] Aqui é mostrada uma *f-zone* de largura maior do que 1. Essa situação é detalhada no capítulo de testes.
[13] Ou máxima, no caso de *speed mod* igual 1 para terreno plano e de fácil deslocamento.

impossível, e portanto trata a célula como bloqueada.

Por fim, usando as informações de passagem indicadas, o mapa local de cada agente em movimento é montado à cada passo da simulação do jogo, e o campo potencial respectivo é construído e relaxado, a fim de se gerarem os vetores gradientes descendentes usados posteriormente pelo módulo de movimento das unidades.

4.6.5 Módulos usados e alterações efetuadas

A Tabela 4.1 mostra um resumo dos principais módulos da *Spring* usados para a implementação do planejamento de caminhos BVP. Alguns dos módulos tiveram que ser modificados, em maior ou menor grau, a fim de completar a inclusão do novo planejador. Todas as alterações realizadas estão disponíveis livremente para *download* pela Internet, conforme detalhes de acesso fornecidos na Seção 5.8.1 deste trabalho.

Tabela 4.1: Módulos da Spring usados na implementação.

Módulo	*Arquivo*	*Funcionalidade fornecida e alterações feitas*
Planejador de caminhos nativo	PathPlanner.cpp	O planejador de caminhos nativo gera caminhos no mapa à partir de solicitações de movimento até um objetivo. A referência global para o planejador de caminhos foi atualizada para apontar para o planejador de caminhos BVP. Uma interface genérica de classe foi construída à partir dos métodos públicos da classe do planejador de caminhos nativo, e o novo planejador BVP foi declarado seguindo a interface em questão.
Movimento terrestre das unidades	GroundMoveType.cpp	Responsável pelo movimento das unidades que andam pelo solo, incluindo a solicitação de criação de caminhos, e controle *frame*-a-*frame* do movimento das unidades. A rotina de atualização do movimento foi modificada de forma à calcular a direção da unidade à partir do vetor de direção fornecido pelo novo planejador e seguindo o método proposto. Também foi alterada de forma à permitir a ativação/desativação do controle de direção entre nativo e BVP, bem como do controle do teste de colisão.
Controle do jogo	Game.cpp	A rotina de atualização do passo da simulação foi atualizada com a chamada para o novo planejador de caminhos. Também foi inserida uma chamada para a rotina de atualização periódica da estrutura BVP, responsável por atividades diversas, como levantamento das estatísticas BVP, e verificação do limite de tempo da partida. Na rotina de atualização de *rendering,* foi inserida uma chamada para a rotina de *rendering* do planejador BVP, que será usada para depuração e para verificação dos testes de qualidade de caminho gerado.
Informações sobre o terreno	MapInfo.cpp e MoveMath.cpp	O módulo fornece informações sobre o terreno, incluindo uma lista de objetos presentes em cada célula, inclinação da célula, e custo de travessia, em termos de *speed mod*.
Manipulador de comandos	Console.h	O módulo permite que seja feita uma configuração de comandos textuais que podem ser digitados no console de jogo, ou então associados a teclas de função ou controle.
Gerenciamento de eventos	EventHandler.cpp	Responsável pelo despacho de eventos de jogo. Alguns dos eventos capturados foram o evento de "Fim de Jogo", para geração de estatísticas, e "Unidade Destruída", para tratamento do desenho do caminho percorrido pela unidade.

Além da modificação sofrida por alguns módulos nativos, novos módulos foram criados para controlar os aspectos da simulação referentes ao planejamento BVP. Os módulos alterados ou criados envolveram não só a inclusão do novo planejador de caminhos BVP, como também englobaram a criação de módulos extras de monitoramento de estatísticas de jogo, recebimento de comandos textuais, geração visual de caminhos, e planejamento configurável por times, necessários para a execução dos testes e levantamento de resultados presentes no capítulo de testes. A Tabela 4.2 mostra uma lista descrevendo os principais módulos criados para uso geral pelo planejamento BVP.

Tabela 4.2: Módulos do planejamento BVP.

Módulo	*Arquivo*	*Funcionalidade*
Controle da estrutura BVP	BVP.cpp	Centraliza o controle das estruturas envolvidas com o funcionamento do planejador BVP, armazenando também a lista de configurações e parâmetros modificáveis do planejador.
Manipulador de comandos	BVPHandler.cpp	Recebimento dos comandos textuais de controle do módulo BVP, permitindo que a configuração interna do planejador BVP seja alterada em tempo de execução.
Estatísticas	BVPStats.cpp	O módulo é responsável pelo levantamento e armazenamento das estatísticas colhidas durante a simulação, o que inclui o tempo de *planning*, *rendering*, número de unidades em movimento, e FPS.
Planejador de caminhos por BVP por times	TeamPathManager.cpp	O módulo simplesmente repassa as requisições de caminho para o planejador nativo ou planejador BVP conforme configurações que podem ser informadas por time. Todos os módulos da *Spring* foram configurados para requisitar caminhos através deste módulo, o que possibilitou a disputa de partidas em que um time utiliza planejador nativo, e outro time utilizava planejador BVP. O módulo também é o responsável por guardar o caminho percorrido pelas unidades, que será usado para os testes de qualidade no capítulo de testes.
Planejador de caminhos BVP	BVPPathManager.cpp	Em analogia ao planejador de caminhos nativo, o módulo recebe requisições de caminho até um objetivo, e fornece como resposta uma estrutura de caminho BVP, que será usada pela unidade para selecionar o caminho a ser percorrido.
Caminhamento BVP	BVPPath.cpp	Mantém uma estrutura de campo potencial representando o mapa local do agente, estrutura de mapa global, e informações sobre o terreno (BVP). Usando o planejamento BVP proposto, fornece a direção e próximo ponto a ser visitado pela unidade durante o caminho até um objetivo.
Campo potencial	PPPotentialField.cpp	Representa o campo potencial do mapa local do agente e todas as rotinas associadas, incluindo a rotina de relaxamento do campo potencial.
Estrutura de mapa global	BVPGlobalMap.cpp	Tem como objetivo fornecer o vetor gradiente global de direção a ser usado pelo mapa local do agente para projetar o objetivo intermediário. Pode ser configurado para gerar o vetor tanto usando o planejador nativo, quanto calculando à partir da posição atual da unidade e posição do objetivo.
Informações sobre o terreno (BVP)	PPTerrainInfo.cpp	Fornece informações sobre o terreno, semelhante às informações fornecidas pelo módulo nativo de informações sobre o terreno. A principal modificação é a possibilidade de configuração se as unidades paradas são consideradas obstáculos[14].

4.6.6 Precisão dos valores de ponto flutuante

A *Spring Engine* utiliza uma biblioteca externa para tratamento dos cálculos de ponto flutuante, chamada de *Streflop*, a fim de proporcionar cálculos de ponto flutuante com reprodutibilidade garantida entre as diversas máquinas e plataformas suportadas pelo jogo, e dessa forma permitir o sincronismo de cálculo necessário entre as diversas simulações distribuídas de uma sessão de jogo *multiplayer*.

Conforme a documentação da *Streflop* (STR 2010), a utilização da biblioteca envolve a remoção total da biblioteca matemática padrão da linguagem C, fazendo com que todas as chamadas referentes à cálculos de ponto flutuante sejam feitas via biblioteca *Streflop*. À título de otimizações de projeto, a *Spring* também foi configurada pelos desenvolvedores para remover, via biblioteca *Streflop*, o suporte ao tipo de dados de ponto flutuante de precisão dupla (*double*, 64 bits) da linguagem C++, mantendo apenas as rotinas de ponto flutuante de precisão simples (*float*, 32 bits). Para completar o ajuste, a *Spring* utiliza chamadas de sistema especiais para forçar a precisão do processador de ponto flutuante para a precisão simples, fazendo automaticamente com que a utilização de valores de precisão dupla seja executada com precisão simples.

Dessa forma, caso seja necessário ser feito algum cálculo que necessite do tipo de dados *double* na *Spring*, deve ser chamada a rotina da biblioteca *Streflop* para ajuste da precisão do

[14] Uma descrição dessa situação é encontrada no capítulo de testes.

processador de ponto flutuante para dupla precisão. Para se manter a correta operação da *Spring*, ao final dos cálculos de ponto flutuante, deve ser chamada a rotina de ajuste de precisão para precisão simples.

Não obstante a necessidade das chamadas indicadas, a versão de precisão dupla das funções padrão para cálculos matemáticos não está disponível na *Spring*, gerando um erro de tempo de montagem de arquivo executável, acusando a ocorrência de identificadores de função indefinidos. Assim, caso seja necessário o uso de funções matemáticas de precisão dupla, como alternativa podem ser utilizadas as rotinas de precisão simples, e caso as mesmas não forneçam a precisão de resultado esperada, deverá ser feita uma adaptação no projeto da *Spring* para que volte a disponibilizar nativamente o tipo de dados e as rotinas de dupla precisão da biblioteca *Streflop*.

5 RESULTADOS EXPERIMENTAIS

A seguir são mostrados os testes executados, o objetivo de cada um, e as configurações utilizadas para a sua realização, acompanhados por gráficos e tabelas apresentando os resultados mais relevantes ou expressivos. Ao final do capítulo são mostradas algumas propostas de modificação ou otimização da implementação do algoritmo, juntamente com alguns resultados obtidos à partir das modificações.

5.1 Objetivo dos testes

O principal objetivo dos testes realizados neste trabalho foi fornecer uma comparação entre o planejador de caminhos nativo da *Spring* e o planejador BVP implementado conforme proposto em (SIL 2009). Três tipos de testes foram realizados:

- Testes de desempenho: tinham como objetivo comparar o planejador nativo e o planejador BVP em termos de escalabilidade, velocidade, limites e desempenho.
- Testes de qualidade: visam permitir uma verificação visual e comparação dos caminhos gerados pelos dois tipos de planejador em algumas situações pré-definidas, a fim de apurar qual dos planejadores apresenta os caminhos mais naturais e menos artificiais.
- Testes de vantagem estratégica: nos testes de vantagem estratégica foram feitas disputas em jogo entre times usando os dois tipos de planejadores, com o objetivo de se verificar se algum dos planejadores ou de suas configurações apresenta algum tipo de favorecimento estratégico que leve o time a uma vitória mais fácil contra o oponente usando a outra configuração.

5.2 Metodologia

Os testes foram executados em mapas de tamanhos variados, bem como com configurações diferentes de parâmetros do planejador BVP, a fim de fornecer um ambiente semelhante para teste dos dois planejadores diferentes, de modo a permitir que fossem feitas comparações entre os resultados.

Nas seções seguintes são mostrados os principais parâmetros possíveis de configurações do planejador, juntamente com uma explicação sobre o método de determinação dos valores iniciais a serem usados para cada parâmetro relevante. Os parâmetros menos relevantes para a realização dos testes foram mantidos com valores fixos em todas as execuções, enquanto que os parâmetros mais importantes foram usados para a montagem de 5 perfis de configuração

detalhados no início da Seção de Teste de Desempenho.

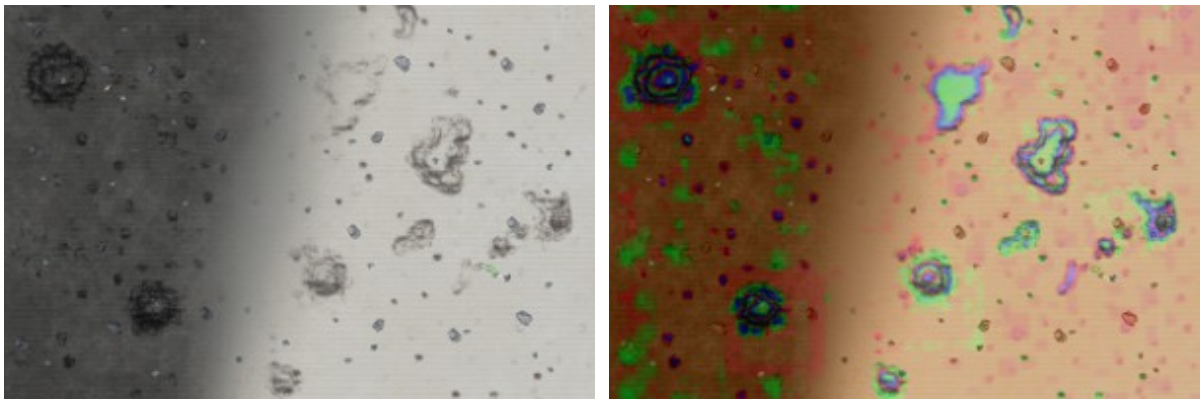
5.2.1 Mapas

Os mapas usados nos testes foram escolhidos conforme o tamanho e dificuldade do terreno, de forma a abranger desde um mapa pequeno e com poucos obstáculos, em que a ação transcorre com poucas unidades, até mapas grandes ou com muitos obstáculos, em que cada partida durará mais tempo ou contará com mais unidades em jogo. Informações sobre o tamanho e quantidade estática de obstáculos pode ser visto na Tabela 5.1.

Tabela 5.1: Tamanho dos mapas usados nos testes.

Mapa	*Grid de células*			*Passáveis*		*Bloqueadas*	
	Largura	*Altura*	*Células*	*Total*	*%*	*Total*	*%*
1 - Darkside Remake	384	256	98.304	95.454	97,10%	2.850	2,90%
2 - Caucasus Skirmish	384	384	147.456	141.559	96,00%	5.897	4,00%
3 - Alien Desert	768	512	393.216	383.213	97,46%	10.003	2,54%
4 - Nuclear Winter	1.280	768	983.040	954.188	97,07%	28.852	2,93%
5 - Road to Rome	1.152	1.152	1.327.104	1.205.164	90,81%	121.940	9,19%

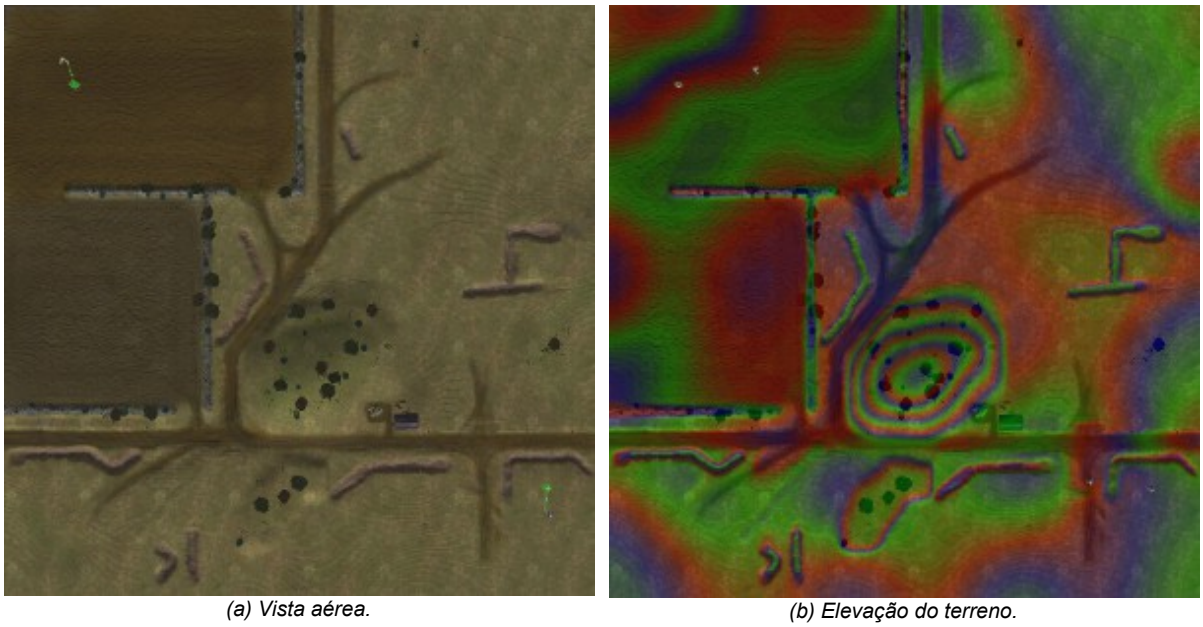
- *Darkside Remake*: O mapa é bem pequeno e praticamente sem obstáculos no terreno. A vista aérea e de elevação do terreno do mapa podem ser encontradas nas Figuras 5.1a e 5.1b, respectivamente.



(a) Vista aérea. *(b) Elevação do terreno.*

Figura 5.1: Mapa "Darkside Remake".

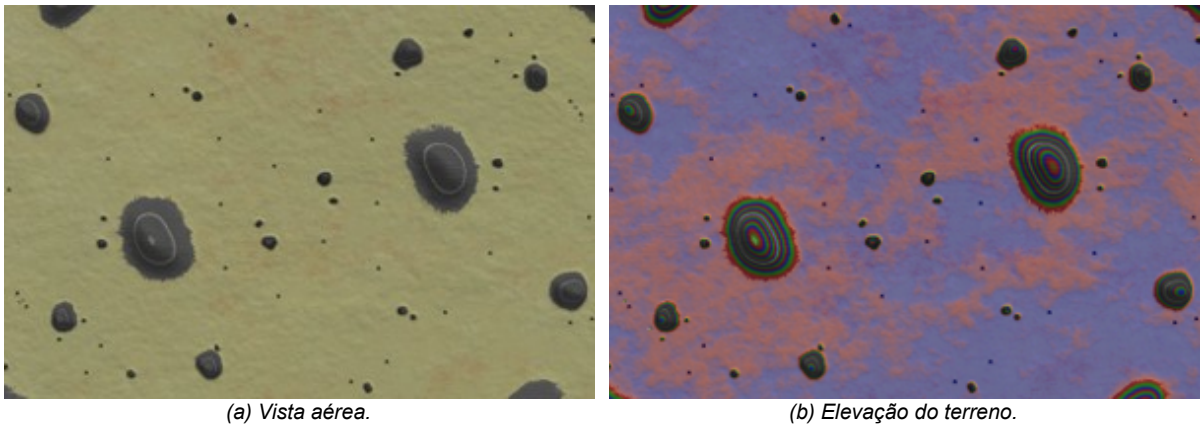
- *Caucasus Skirmish*: O mapa é pequeno, mas o terreno possui muitas variação de elevação, com uma montanha ao centro. A Figura 5.2a mostra a vista aérea do mapa, e a Figura 5.2b mostra a informação de elevação do terreno.



(a) Vista aérea. *(b) Elevação do terreno.*

Figura 5.2: Mapa "Caucasus Skirmish".

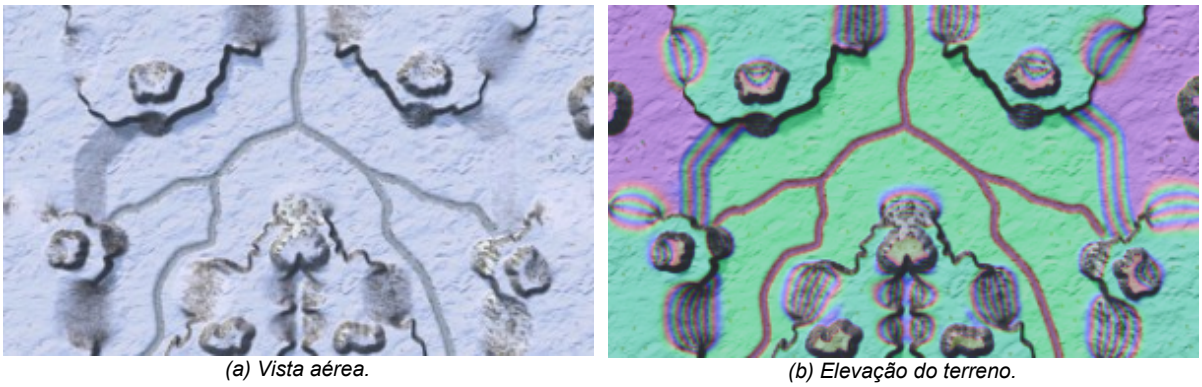
- *Alien Desert*: O terreno é plano, com alguns obstáculos grandes que devem ser contornados pelas unidades. Informações sobre a vista aérea e elevação do terreno do mapa podem ser vistas, respectivamente, nas Figuras 5.3a e 5.3b.



(a) Vista aérea. *(b) Elevação do terreno.*

Figura 5.3: Mapa "Alien Desert".

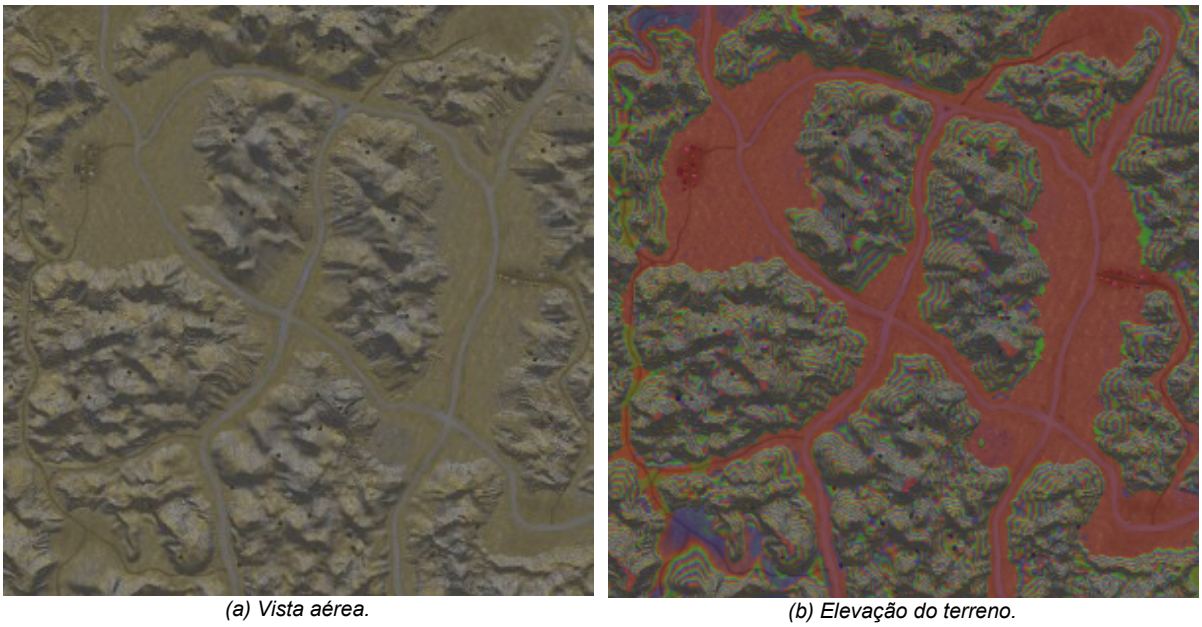
- *Nuclear Winter*: o mapa é grande, e o terreno bem variado, contendo planícies, montanhas, penhascos e paredões. A vista aérea e de elevação do terreno do mapa podem ser encontradas nas Figuras 5.4a e 5.4b, respectivamente.



(a) Vista aérea. (b) Elevação do terreno.

Figura 5.4: Mapa "Nuclear Winter".

- *Road to Rome*: As estradas correm na parte plana do mapa, que possui muitas montanhas, onde o terreno é difícil. É o maior dos mapas testados, e o que possui mais obstáculos. As informações sobre a vista aérea e elevação do terreno do mapa podem ser vistas, respectivamente, nas Figuras 5.5a e 5.5b.



(a) Vista aérea. (b) Elevação do terreno.

Figura 5.5: Mapa "Road to Rome".

5.2.2 Configurações

O algoritmo planejador BVP admite um certo nível de configuração em alguns dos parâmetros de uso.

- Precisão numérica do campo potencial: a precisão numérica afeta diretamente a

capacidade de propagação do potencial do objetivo intermediário, e deve ser levada em conta no cálculo do campo potencial.

- Largura da borda em *f-zone* no mapa local do agente: o objetivo da *f-zone* é garantir a propagação da informação do objetivo intermediário até a posição do agente. Como foi explicado em Silveira *et al.*, o problema de planificação pode ocorrer em situações em que regiões estejam separadas por passagens muito estreitas, o que também vale para a propagação da informação de potencial por uma *f-zone* com largura muito estreita ou insuficiente para realizar uma boa propagação.
- Tamanho do mapa local do agente: conforme proposto em Silveira *et al.*, o tamanho do mapa local é definido por um cone de visualização que representa o alcance dos sensores do agente. Além disso, quanto mais informação do ambiente estiver disponível para o agente, mais predisposição ele terá de modificar o seu comportamento para evitar regiões desordenadamente cheias de obstáculos.
- Tamanho e posição da célula de objetivo intermediário no mapa local do agente: o tamanho e posição exata do objetivo intermediário também é relevante para o problema da planificação no mapa local, já que influencia na propagação da informação de movimento do agente.
- Vetor gradiente global: o vetor gradiente global serve para indicar a posição em que será mapeado o objetivo intermediário na borda do mapa local do agente, podendo ser fornecido não só por um mapa global, mas também de outras maneiras simplificadas.
- Quantidade de relaxamentos a cada passo: cada iteração do processo de relaxamento propaga um pouco mais o valor de potencial do objetivo para as outras células. Quantidades insuficientes de relaxamento talvez não propaguem o valor de potencial até o centro do mapa, onde se encontra o agente no caso do mapa local proposto.
- Vetor de comportamento: a direção e influência do vetor de comportamento podem ser ajustadas a fim de se obter um caminho único e pessoal do agente, de forma a aumentar o realismo dos caminhos gerados.

Além das configurações permitidas pelo planejador BVP, a implementação do método na *Spring* resultou nas seguintes possibilidades de configuração dos parâmetros de movimento das unidades:

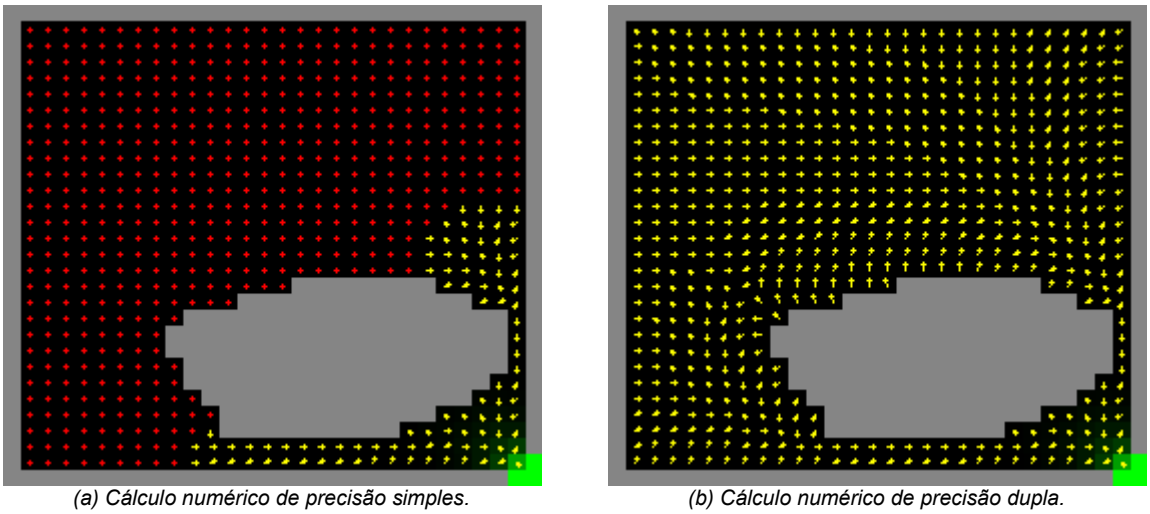
- Verificação de colisão: a verificação de colisão entre as unidades impede que duas unidades ocupem o mesmo local na simulação ao mesmo tempo, através da simulação de colisão entre as unidades. O módulo de verificação de colisão precisou ser desativado para a utilização do planejador BVP, porém seu uso pode ser configurado para fins de teste.
- Controle de direção: as unidades já possuem um controle de direção, também desativado para o uso do planejador BVP. Contudo, o controle de direção nativo pode ser utilizado à partir da informação de direção fornecida pelo gradiente descendente do mapa local do agente.
- Bloqueio por unidades paradas: a inteligência artificial do oponente tende a

construir estruturas e posicionar as unidades trabalhadoras de forma muito próxima ou compacta dentro da base. Em determinadas situações, as unidades podem estar tão próximas que pode não haver uma saída possível em alguns pontos da base, como por exemplo de dentro das fábricas que produzem unidades. O planejador de caminhos nativo resolve bem esta situação fazendo com que as unidades bloqueadas empurrem as unidades paradas, abrindo passagem dessa forma.

5.3 Levantamento das configurações iniciais

5.3.1 Precisão numérica do campo potencial

Como já foi mostrado anteriormente, a precisão numérica afeta a qualidade de propagação do potencial do objetivo intermediário para os outros pontos do campo potencial. A Figura 5.6 mostra um exemplo de mapa local do agente em que o objetivo está mapeado atrás de um grande obstáculo. As flechas amarelas representam o gradiente descendente de cada célula, e as cruzes vermelhas mostram regiões de planificação, onde não foi possível obter o vetor gradiente. As respectivas diferenças no campo potencial gerado conforme a precisão adotada no cálculo da propagação do potencial podem ser acompanhadas nas Figuras 5.6a e 5.6b.



(a) Cálculo numérico de precisão simples. *(b) Cálculo numérico de precisão dupla.*

Figura 5.6: Influência da precisão numérica no campo potencial.

A baixa qualidade de propagação apresentada pelo cálculo usando precisão simples pode afetar diretamente o desempenho da simulação, já que serão necessárias mais iterações no processo de relaxamento para propagar o potencial do objetivo intermediário para todos os pontos do campo quando da ocorrência de situações de pior caso conforme o exemplo da figura, ou caso contrário as unidades podem ficar trancadas devido ao problema da planificação.

Todavia, a utilização de valores numéricos de precisão dupla ou invés de simples também causa um impacto negativo no desempenho do processo de relaxamento, já que será usado o dobro da memória necessária pelo cálculo de precisão simples, e também será exigida a execução de instruções de processamento que necessitam de mais precisão, e que podem ser relativamente mais lentas dependendo do *hardware* disponível para execução do código.

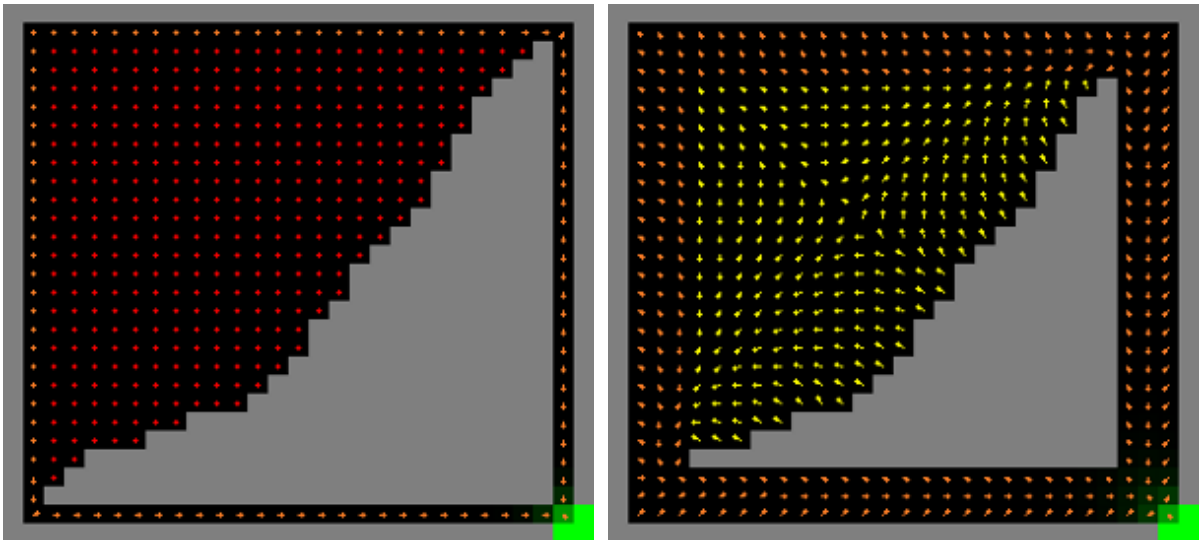
Independente do impacto causado pela precisão de cálculo escolhida, todos os testes foram realizados usando o cálculo de relaxamento do campo potencial com precisão dupla, pois nos muitos testes preliminares executados, o cálculo de precisão simples resultou em uma frequência alta de unidades trancadas devido à planificação, o que não ocorreu quando do uso de precisão dupla, afetando de maneira severa os resultados dos testes, e obscurecendo a verificação dos outros detalhes a serem investigados.

5.3.2 Tamanho da borda *f-zone* do mapa local do agente

A Figura 5.7a mostra um mapa local de agente de tamanho 30x30[15], com uma situação que poderia representar um pior caso, em que o objetivo se encontra atrás de um grande obstáculo. No caso em questão, a borda estreita da região de *f-zone* (pontos com cor laranja) impediu totalmente que o potencial do objetivo intermediário fosse propagado até o centro do mapa, onde se encontra o agente, e onde será obtido o vetor gradiente descendente para cálculo da direção do agente.

A Figura 5.7b mostra o mesmo exemplo de situação da Figura 5.7a, porém para o cálculo do mapa local foi usada uma borda de tamanho 3 para a região *f-zone*. Nesse caso, foi possível propagar o valor de potencial do objetivo intermediário até a região central do mapa.

Para a execução dos testes, à menos que informado em contrário, foi usado o tamanho de borda *f-zone* de largura 3.



(a) Boda de f-zone de tamanho 1. *(b) Borda de f-zone de tamanho 3.*

Figura 5.7: Diferenças no tamanho da borda de f-zone.

5.3.3 Tamanho do mapa local do agente

O tamanho do mapa local do agente é um valor muito subjetivo para uso na simulação. Nos exemplos utilizados em (SIL 2009), são usados mapas quadrados variando entre 15 e 55 de largura.

Durante alguns testes preliminares para levantamento dos parâmetros a serem utilizados,

[15] Tamanho arbitrário.

valores pequenos para o tamanho do mapa, por exemplo entre 10 e 20, resultaram em menos cálculo no processo de relaxamento, possibilitando que mais agentes se movessem ao mesmo tempo sem que a simulação sofresse uma perda grande de velocidade. Contudo, o mapa local de tamanho pequeno também não ajudou muito as unidades à contornarem obstáculos grandes, fazendo com que ficassem presas atrás de concavidades, ou até mesmo detectassem tardiamente os obstáculos devido ao curto alcance do sensor.

Quanto aos mapas locais de tamanho grande, entre 40 e 50, levaram a um melhor tratamento e desvio de obstáculos das unidades. Porém também resultaram em uma maior ocorrência de planificação na presença de muitos obstáculos, ou que tenham um tamanho grande. A ocorrência de planificação resultou em unidades que permaneceram paradas, ou então girando em torno do próprio eixo, sem tomar uma direção.

A Figura 5.9 usada na explicação da seção seguinte mostra um mapa de tamanho 30x30 que já possui uma área de planificação pequena em um dos cantos. Á medida que o tamanho do mapa local e a quantidade de obstáculos cresce, mais próximo do centro do mapa local estará a área de planificação.

Um tamanho de mapa local razoável para uso na *Spring* foi encontrado experimentalmente com tamanho de 29x29. Considerando o número ímpar do tamanho de forma que se possa posicionar a unidade exatamente no centro, uma borda não utilizável de largura 1 para a *b-zone* e 3 para a *f-zone*, e um tamanho típico de unidade de 4x4, sobram entre 8 e 10 células de sensoriamento em cada direção ortogonal da unidade. Um exemplo de mapa local usando as medidas informadas pode ser visto na Figura 5.8.

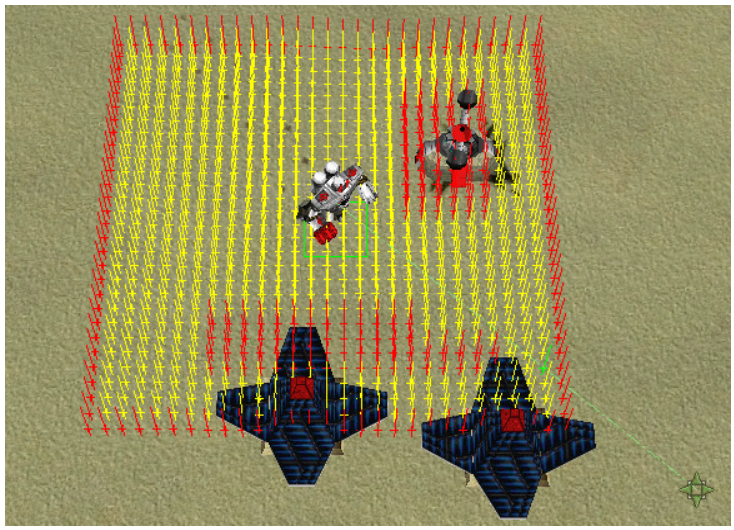


Figura 5.8: Exemplo de mapa local da unidade.

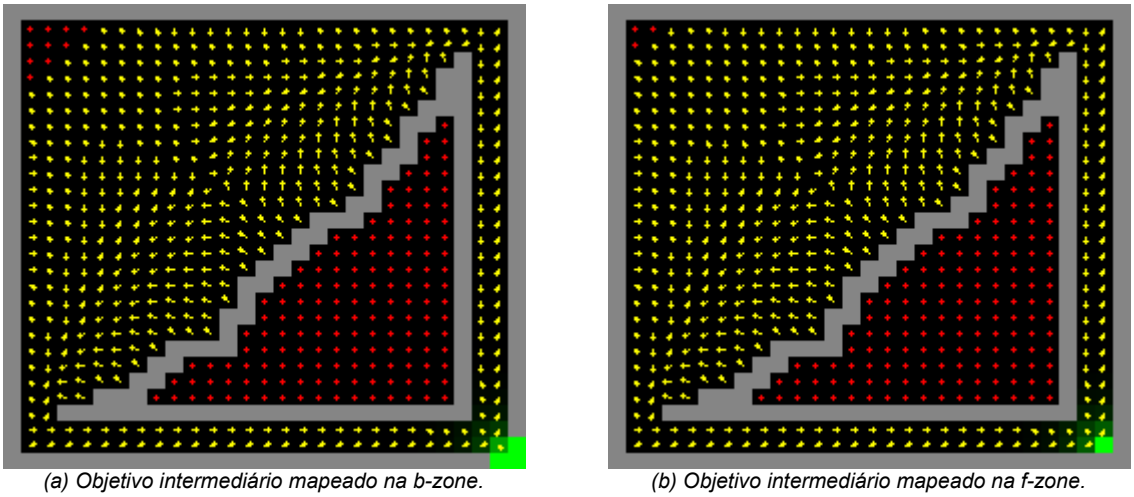
5.3.4 Mapeamento do objetivo intermediário

Conforme proposto em (SIL 2009), o objetivo intermediário deve ser mapeado na borda do mapa local do agente, de forma que o potencial do objetivo seja propagado via a região *f-zone* à todos os pontos do campo potencial, incluindo a região central onde estará localizado o agente.

O método proposto mostra o objetivo como sendo 3 pontos mapeados dentro da borda do mapa, na região de *b-zone*. Alguns testes experimentais sintéticos e também dentro do jogo indicaram que o ponto intermediário pode gerar a mesma qualidade de propagação sendo somente 1 ponto e mapeado dentro da *f-zone*. A modificação facilita a programação da rotina

de projeção do objetivo na borda do mapa, além de poupar algumas instruções, o que pode contribuir para o aumento do desempenho da simulação. A Figura 5.8 mostrada anteriormente apresenta o objetivo intermediário como apenas 1 ponto (em verde no mapa) mapeado dentro da *f-zone*. As Figuras 5.9a e 5.9b mostram uma comparação entre uma possível situação de pior caso, com obstáculo muito extenso, e a propagação resultante do objetivo intermediário tanto mapeado na região *b-zone* (Figura 5.9a), como na região *f-zone* (Figura 5.9b). No caso do objetivo mapeado na *f-zone*, a propagação atinge até valores mais distantes de células.

Para a execução dos testes, e à menos que informado em contrário, o objetivo intermediário foi mapeado como 1 ponto e dentro da *f-zone*.



(a) Objetivo intermediário mapeado na b-zone. *(b) Objetivo intermediário mapeado na f-zone.*

Figura 5.9: Mapeamento do objetivo intermediário.

5.3.5 Vetor gradiente global

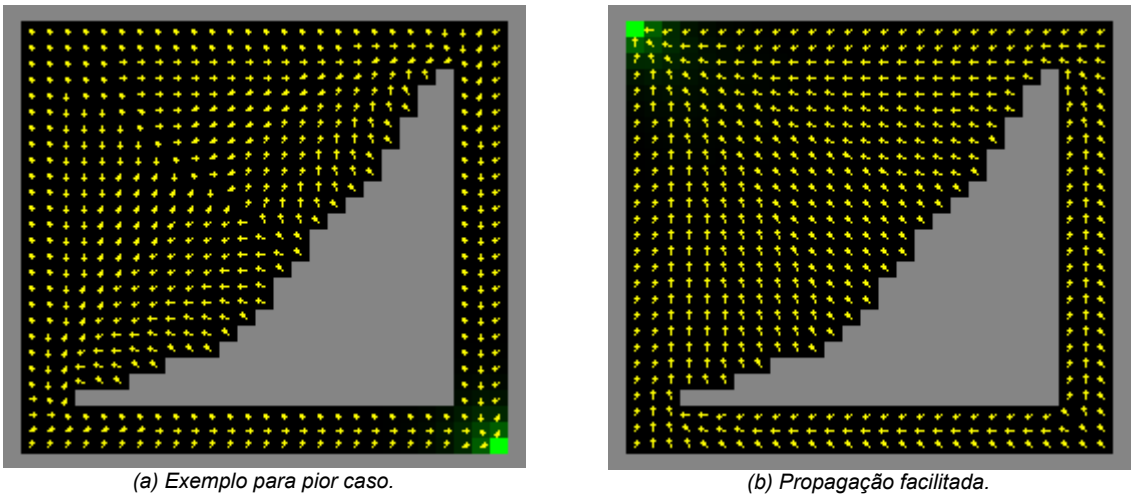
Considerando que o único objetivo do mapa global é o fornecimento do vetor gradiente de direção que servirá para mapear o objetivo intermediário dentro do campo potencial do mapa local do agente, e considerando também que a *Spring* já possui um planejador de caminhos funcional e que já leva em conta também os custos de travessia do terreno, a rotina de cálculo do vetor gradiente global foi construída de forma a calcular o vetor de direção usando o planejador de caminhos nativo da *Spring*. Construída dessa forma, o gradiente global levará em conta o custo do terreno, fazendo com que a unidade evite caminhos que levem a obstáculos muito grandes, ou terrenos com custo muito alto, uma informação que o mapa local normalmente não pode fornecer, mesmo usando um mapa global baseado em campo potencial.

Todavia, o vetor gradiente global também pode ser calculado sem um mapa global, de forma que a direção seja dada pela direção da unidade em relação ao objetivo. Dessa forma, o objetivo intermediário sempre será mapeado na direção do objetivo final, e a evasão de obstáculos ficará totalmente a cargo do mapa local do agente.

5.3.6 Quantidade de relaxamentos

Durante os experimentos para determinar os valores iniciais dos testes, foi arbitrado um valor máximo de 100 relaxamentos. Nas análises de pior caso com obstáculo grande situado nas bordas e com o objetivo intermediário mapeado exatamente atrás do obstáculo (Figura

5.10a), valores de relaxamento à partir de 72 resultaram em um campo totalmente livre de planificação. Na análise de um caso com uma propagação facilitada, como na Figura 5.10b, valores bem baixos, em torno de 25, resultaram em um campo livre de planificação.



(a) Exemplo para pior caso.

(b) Propagação facilitada.

Figura 5.10: Influência da posição do objetivo intermediário e obstáculos.

Cada iteração do relaxamento possui um alto custo computacional, que envolve diversas operações de ponto flutuante. Uma diferença de quase 3 vezes no número de relaxamentos entre 25 e 72, como mostrada no exemplo, poderia levar a uma lentidão e desperdício do tempo computacional, no caso de relaxamentos desnecessários.

Fischer (FIS 2008, p.61) mostra um exemplo de rotina de relaxamento que usa um valor limite de "erro acumulado" para limitar o número de iterações do laço de relaxamento. O "erro acumulado" é considerado como sendo a soma das diferenças absolutas das variações de potencial entre a iteração atual e a iteração anterior. No exemplo de código apresentado por Fischer, quando o "erro acumulado" ficar abaixo de um valor mínimo pré-determinado, o laço de relaxamento é encerrado, considerando dessa forma que o objetivo de relaxamento foi atingido e que o campo potencial convergiu.

Seguindo a mesma ideia, a Tabela 5.2 mostra uma análise da ordem de grandeza do erro acumulado nos dois testes efetuados anteriormente.

Tabela 5.2: Erro acumulado no relaxamento.

Situação	*Relaxamentos*	*Erro acumulado*
1 - propagação facilitada	25	0,03738
	72	0,009504
2 - exemplo do pior caso	25	7,8796E-4
	72	1,1449E-8

Considerando os extremos da faixa de erro acumulado mostrado na tabela, a diferença entre o maior valor e o menor valor é de aproximadamente 33 milhões de vezes. Essa diferença tão grande entre as situações distintas não ajudou a uniformizar o corte do número de relaxamentos, já que a escolha de um valor muito pequeno, perto do valor da situação 2, resultou na ocorrência de um número extra desnecessário de relaxamentos na situação 1. Já a

escolha de um número alto para o limite do erro acumulado causou um encerramento prematuro e insuficiente do relaxamento em casos semelhantes à situação 2. A Figura 5.11 mostra um exemplo de relaxamento da situação 2 com limite de erro ajustado para o valor de erro acumulado obtido da situação 1 (0,03738).

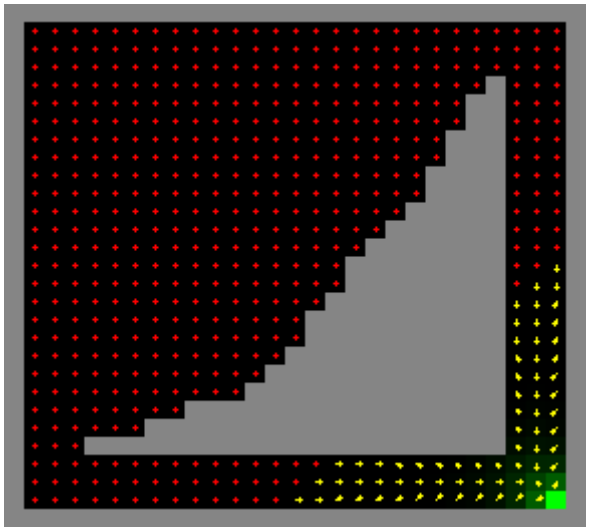


Figura 5.11: Corte prematuro no relaxamento por erro acumulado.

No exemplo, foram executados apenas 12 relaxamentos, e após isso o erro acumulado caiu abaixo do valor de limite ajustado para 0,03738. Isso mostra que o recurso do erro acumulado, por si só, não ajuda a diminuir o número de relaxamentos desnecessários quando for necessária uma garantia maior de que não ocorra planificação nos mapas locais.

Em vista disso, nos testes realizados, a menos que disposto em contrário, foi usado um valor de limite de relaxamento de 100, e um valor de limite de erro acumulado ajustado para 0.00. O valor de limite de relaxamento de 100, acima dos valores de teste de 72 mostrados, garantiu uma boa margem para propagação na ocorrência de obstáculos muito intrincados. O valor de erro ajustado para 0.0 garantiu que não sejam feitos cortes prematuros no relaxamento.

5.3.7 Vetor de comportamento

Conforme proposto em (SIL 2009), o vetor de comportamento e seu respectivo fator de influência devem ser ajustados experimentalmente a fim de se obter o comportamento desejado. Para manter uma padronização e uniformização dos resultados calculados, nos testes o fator de influência do vetor de comportamento foi mantido em zero, o que, na prática, cancelou a utilização do vetor de comportamento, e forneceu resultados padrão independentes de comportamento configurado.

5.3.8 Verificação de colisão

Como a verificação de colisão nativa pode ser ativada ou desativada conforme necessário, alguns dos testes foram feitos usando a configuração ativada, a fim de verificar o impacto da configuração nos resultados.

5.3.9 Controle de direção

O controle de direção gerencia a maneira como as unidades fazem curvas, viram, caminham, correm, aceleram e param. Da mesma forma como a verificação de colisão, alguns dos testes foram feitos usando a configuração nativa do controle de direção, a fim de se verificar eventuais diferenças nos resultados obtidos.

5.3.10 Bloqueio por unidades paradas

Como informado anteriormente, durante alguns testes preliminares ocorreu uma situação não prevista pelo planejador BVP, que é a de ocorrência de unidades trabalhadoras trancando a saída das fábricas, ou de obstáculos muito próximos em que não haja saída desbloqueada. Para levar essa situação em conta nos testes, alguns dos testes trataram as unidades paradas como caminho livre no mapa local das unidades, e a verificação de colisão da *Spring* gerenciou a colisão e abertura de caminho através de empurrões entre as unidades, da mesma forma como já funciona no planejador de caminhos nativo.

5.4 Desempenho

Os testes de desempenho têm como principal objetivo mostrar uma comparação de escalabilidade entre o planejador de caminhos nativo da *Spring*, e o planejador de caminhos BVP.

As variações de configuração de planejamento usadas nos testes podem ser vistas na Tabela 5.3.

Tabela 5.3: Variações de configuração usadas nos testes de desempenho.

Configuração	*Controle de direção*	*Mapa global*	*Verificação de colisões*	*Unidades paradas*	*Descrição*
Nativo	*nativo*	-	ativada	padrão da Spring	Dois oponentes controlados pela IA, usando o planejador nativo.
BVP 1	bvp	nativo	desativada	tratadas como obstáculo	Dois oponentes controlados pela IA, usando o planejador BVP na íntegra. A verificação de colisões está desativada, e as unidades paradas são tratadas como obstáculos.
BVP 2	bvp	nativo	ativada	passagem livre	Dois oponentes controlados pela IA, usando o planejador BVP. A verificação de colisões está ativada, e as unidades paradas são tratadas como células livres.
BVP 3	bvp	simplificado	ativada	passagem livre	Dois oponentes controlados pela IA, usando o planejador BVP, com mapa global simplificado dado pela direção da unidade ao objetivo.
BVP 4	nativo + mapa local	nativo	ativada	passagem livre	Dois oponentes controlados pela IA, com controle de direção nativo usando a direção fornecida pelo mapa local da unidade.

A Tabela 5.4 contém o resumo das configurações fixas usadas nos testes de desempenho:

Tabela 5.4: Resumo das configurações usadas nos testes de desempenho.

Configuração	*Valor*
Computador	Intel Core 2 Duo 2.40GHz, 2GiB RAM DDR2
Placa de vídeo	NVIDIA GeForce 8500 GT 256MiB

Configuração	*Valor*
Índice de Experiência do Windows	Processador: 5,8 Memória RAM: 5,5 Gráficos de jogos: 5,5 Total: 4,9
Sistema Operacional	Windows 7 Professional 32 bits
Quantidade de mapas	5 mapas
Quantidade de configurações por mapa	5 testes
Quantidade de repetições de cada teste	3 vezes
Quantidade total de testes	75 testes
Limite de tempo de cada teste	30 minutos por teste, ou 37h30 de tempo total de simulação
Condição de final de jogo	Destruição de todas unidades do oponente, ou por limite de tempo
Precisão numérica do campo potencial	ponto flutuante *double* - 64bits
Borda de *f-zone*	largura de 3 células
Objetivo intermediário do mapa local	1 célula, mapeada na *f-zone*
Limite máximo de relaxamentos	100 iterações
Tamanho do mapa local	29x29 células
Limite de erro de relaxamento	(sem limite) 0.00
Vetor de comportamento	não utilizado - influência zero
Limite de *speed mod*	0.01 (padrão da *Spring*)
Sincronismo vertical (VSync)	desativado
Versão da Spring	v.0.82.1.0
Inteligência artificial (BOT)	Kloots Skirmish AI (KAIK) versão 0.13
Mod de jogo	XTA 0.96

Cada teste foi repetido 3 vezes, e os valores foram usados à partir do teste ou testes que mostraram valores mais estáveis ou relevantes, como por exemplo o que durou mais tempo, apresentou o maior número de unidades em jogo, ou terminou o jogo com sucesso antes de ultrapassar o limite de tempo.

Ao final de cada teste, os valores relevantes referentes à simulação foram anotados em um arquivo de texto, para posterior importação em formato de planilha de cálculo. Os gráficos gerados visam principalmente comparar o funcionamento do planejador BVP em relação ao planejador nativo. Os valores de tempo amostrados em milissegundos possuem uma margem de erro da ordem de ±1 milissegundo, seguindo a precisão da rotina de amostragem de tempo disponível na biblioteca SDL usada pela *Spring*. Os valores de números de unidades, a menos que especificado em contrário, representam o número de unidades se movendo, utilizando ativamente o planejamento de caminhos. Resultados obtidos para um número de *frames* muito baixo (menos de 50 *frames* durante toda a simulação) podem se apresentar muito distorcidos e, nesse caso, foram cortados dos gráficos e resultados.

A rotina periódica de atualização da simulação da *Spring* (rotina *SimFrame*, em Game.cpp) foi configurada pelos desenvolvedores para rodar a uma taxa fixa de 30 iterações por segundo, sendo a responsável por todas as atualizações de movimento, *planning*, combate,

física, comandos, e demais detalhes periódicos da simulação, com exceção da parte gráfica. Quanto à rotina periódica de *rendering* (rotina *Draw*, em Game.cpp), responsável pelos gráficos do jogo e demais recursos visuais, está configurada para rodar na máxima velocidade possível[16], com o objetivo de fornecer a máxima taxa de atualização que o computador permitir.

Em vista da taxa de atualização da simulação estar fixa em 30 FPS, as variações no tempo gasto com a rotina são compensadas pelo laço principal de atualização do jogo (rotina *ClientReadNet*, em Game.cpp), mantendo a taxa de atualização fixa e reduzindo o tempo disponível para atualização gráfica. Dessa forma, flutuações no tempo gasto com a atualização da simulação são compensadas pelo laço principal através de uma variação no tempo disponível para a atualização gráfica. Essa forma de implementação das rotinas de atualização de simulação e de gráficos permite a implementação de jogos *multiplayer* em que, facilmente, se mantém um sincronismo entre a simulação dos diversos jogadores, porém permitindo que cada jogador tire o máximo proveito dos seus recursos gráficos disponíveis.

Nos testes, os tempos de *rendering* informados dizem respeito ao tempo médio de execução da rotina *Draw*, dado em milissegundos, e os tempos de *planning* representam o tempo médio de execução da rotina de atualização de planejamento, chamada de dentro da rotina *SimFrame*. A taxa de FPS, quando informada, representa a taxa de atualização gráfica média[17] percebida pelo jogador. Valores isolados em termos de quantidade de *frames* dizem respeito à execução da rotina de atualização de simulação.

Todas estatísticas colhidas para os testes foram amostradas durante a rotina de atualização da simulação. Valores amostrados que ocorrem em frequência maior ou de forma assíncrona em relação à rotina de atualização foram amostrados como valores médios obtidos durante cada *frame* da simulação.

Foi criado um programa para automatizar a execução dos testes, de forma que não houvesse interação humana na execução do programa e gravação dos resultados. Assim, os 75 testes de desempenho rodaram durante 26 horas, 53 minutos, e 33 segundos.

5.4.1 Primeiros resultados: mapa 1

5.4.1.1 Planejador nativo

A Figura 5.12 mostra um gráfico comparativo entre a progressão de tempo do jogo usando o planejador nativo, considerando o FPS e o número de unidades. O objetivo foi estabelecer a base de desempenho para a análise dos resultados. O gráfico mostra que o desempenho se manteve praticamente estável até os 10 minutos de jogo, quando então sofreu uma variação grande, mas que porém não afetou a interatividade da simulação.

[16] Considerando o recurso *VSync* desativo.
[17] Taxa de FPS oficial da *Spring*, e atualizada a cada 1 segundo.

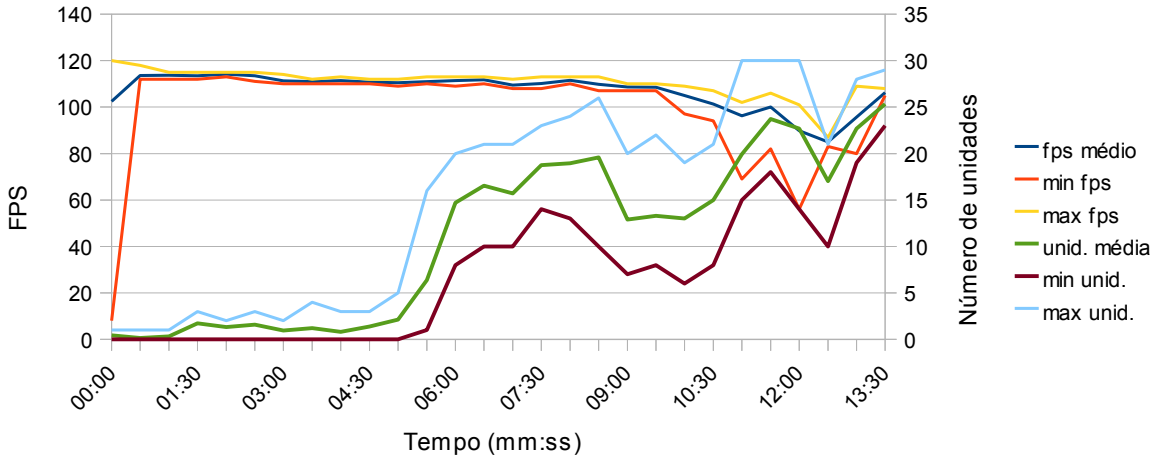


Figura 5.12: Progressão do número de unidades no planejador nativo.

Contudo, o gráfico da Figura 5.12 não deixa claro se foi o aumento do número de unidades que causou uma diminuição do desempenho. O gráfico presente na Figura 5.13 tem como objetivo esclarecer esta dúvida. O gráfico mostra que a diminuição do FPS ocorrida próximo de 10 minutos de simulação foi devido à um aumento brusco do tempo de *rendering*, devido à um possível aumento do número de unidades em jogo. No mesmo momento da queda do desempenho, o tempo médio gasto com *planning* pelo planejador nativo se manteve baixo e estável.

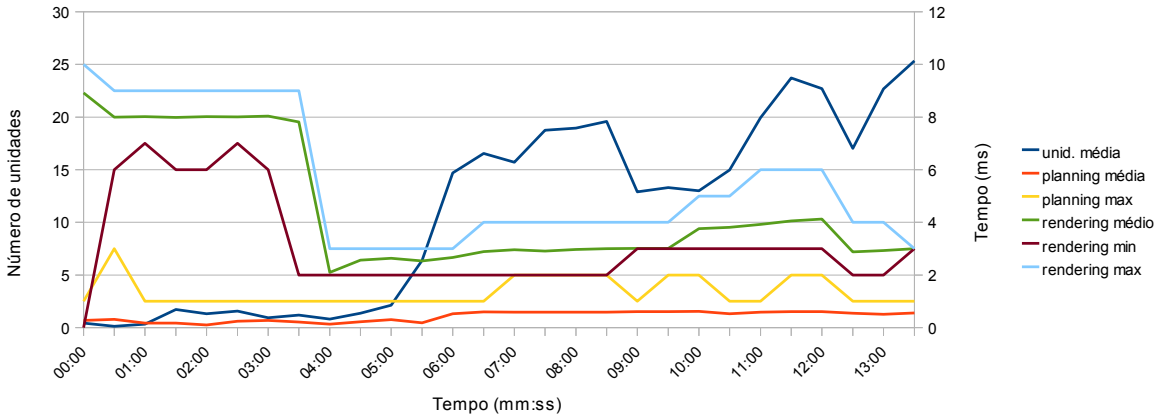


Figura 5.13: Tempos de rendering e planning do planejador nativo.

A Figura 5.14 mostra a distribuição do tempo de simulação, em relação ao número de unidades em movimento usando o planejador nativo no mapa 1. Um maior tempo de *frames* amostrados com determinada quantidade de unidades possibilita uma maior precisão e confiança nos valores a serem apresentados nos gráficos seguintes.

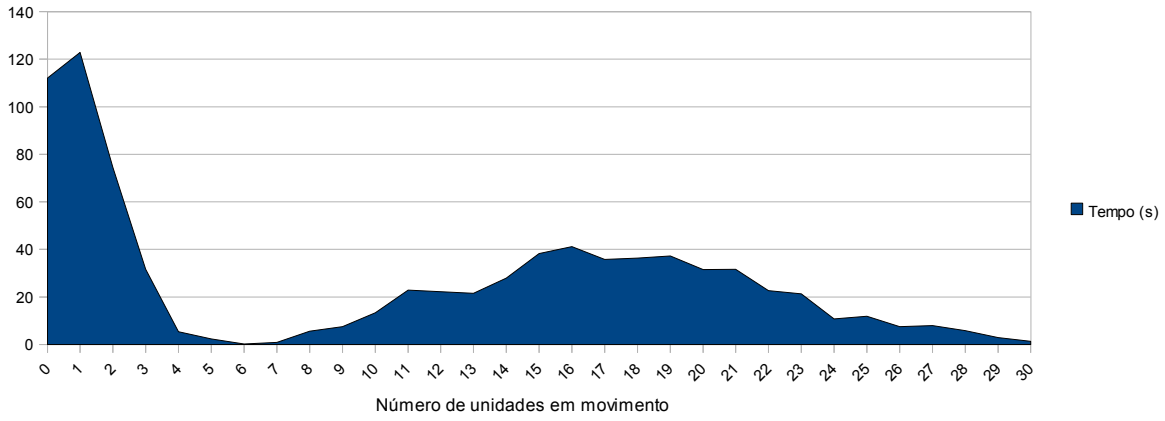


Figura 5.14: Distribuição do tempo de simulação.

A Figura 5.15 mostra um gráfico que relaciona o FPS, tempo de *rendering*[18], e o tempo de *planning*[19], em relação ao número de unidades em movimento controladas pelo planejador nativo. O gráfico mostra um tempo de *planning*, tanto médio quanto máximo, estável em relação ao número de unidades. As variações de desempenho também são poucas, e acompanham as variações no tempo de *rendering*.

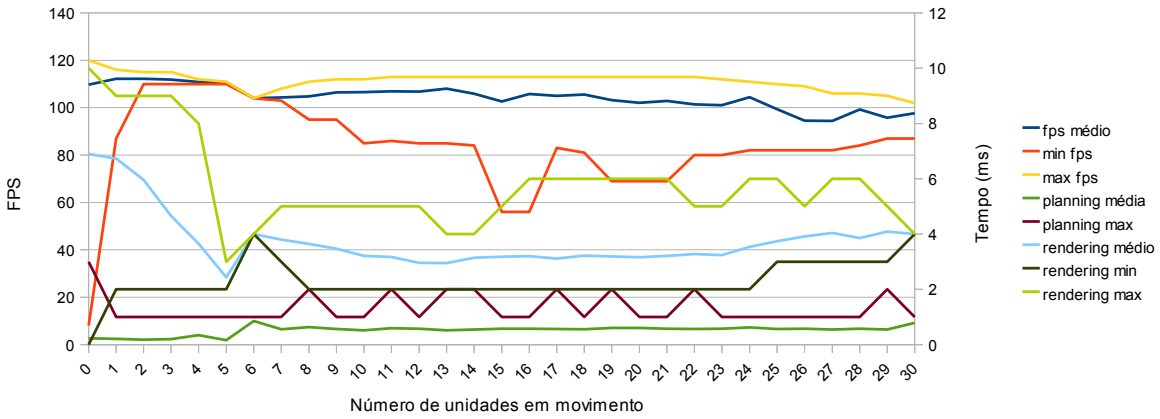


Figura 5.15: Relação entre FPS, tempo de planning e rendering.

A conclusão que se obtém dos gráficos mostrados é de que, no mapa 1, e para um número de unidades em movimento entre 0 e 30, o planejador de caminhos nativo não provoca variação de desempenho relevante durante a simulação do jogo.

5.4.1.2 Planejador BVP

A Figura 5.16 mostra, em um mesmo gráfico, o comparativo de execução entre os 4 tipos de configuração do planejador BVP, relacionando o FPS médio com o número de unidades. Pode-se constatar que as quatro configurações rodaram em velocidade interativa durante uma boa parte da simulação. Entretanto, as configurações 1, 3 e 4 apresentam uma queda forte na taxa de FPS quando o número de unidades passou de 20, sendo que as configurações 1 e 3 apresentaram um FPS abaixo de 10 para valores de unidade acima de 30. Como a configuração nativa não rodou, no mapa em questão, um número grande suficiente de unidades para que pudesse ser feita uma comparação, não se pode afirmar somente através do

[18] Tempo gasto no *frame* desenhando os elementos gráficos da simulação.
[19] Tempo gasto no *frame* com rotinas relacionadas ao planejamento de caminhos.

gráfico em questão que a queda brusca no desempenho foi causada pela utilização das configurações BVP com um número muito grande unidades.

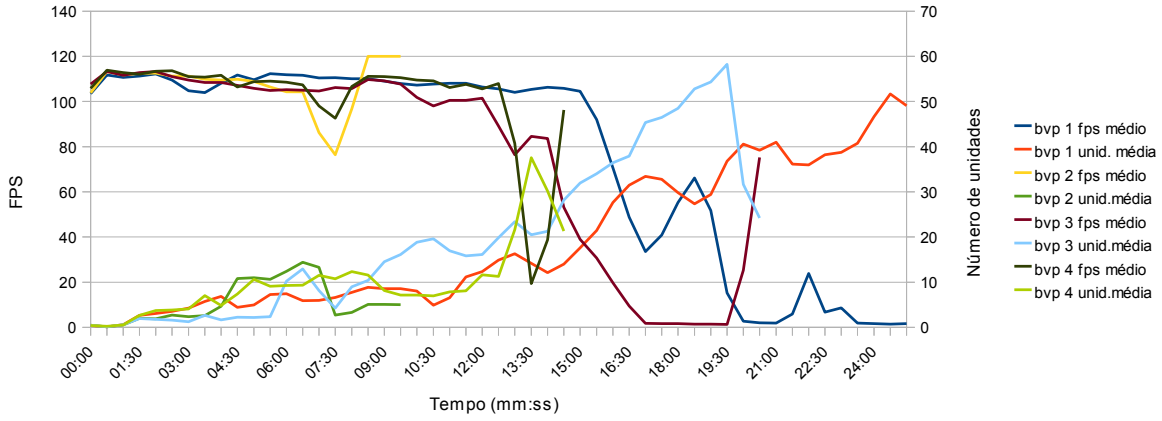


Figura 5.16: Variação entre FPS e número de unidades no planejador BVP.

A fim de esclarecer o motivo das quedas bruscas de desempenho, a Figura 5.17 mostra um gráfico comparando a evolução do tempo de *rendering* e de *planning* do planejador BVP. O gráfico mostra que, enquanto na média o tempo de *rendering* se manteve abaixo de 4ms, o tempo de *planning* sofreu picos de quase 45ms, variando praticamente proporcionalmente ao número de unidades, caso seja feita uma comparação entre os gráficos da Figura 5.16 e Figura 5.17.

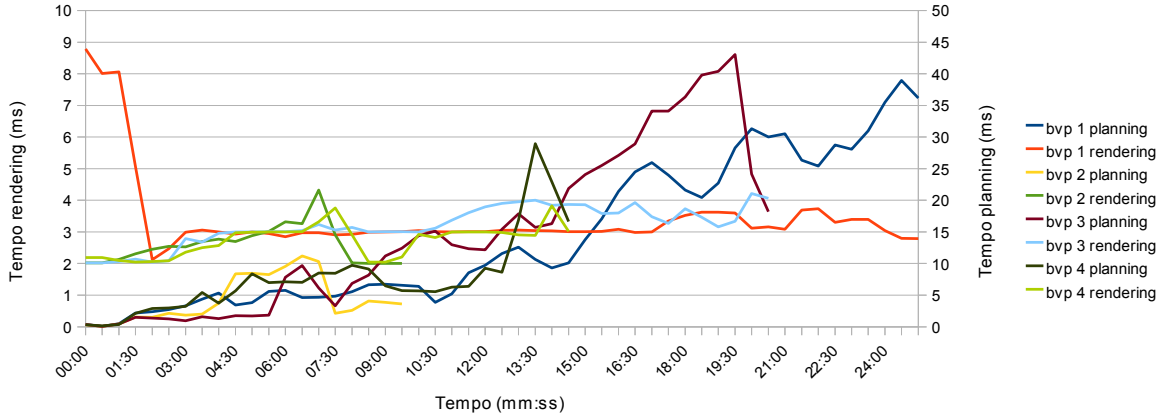


Figura 5.17: Tempo de rendering e planning do planejador BVP.

Em vista disso, o baixo desempenho, abaixo de 10 *frames*-por-segundo, das simulações 1 e 3 foi devido ao grande consumo de tempo de *planning* das simulações, tendo afetado negativamente o desempenho à partir de 30 unidades em movimento.

Aproveitando a Figura 5.18, que mostra o gráfico comparativo de desempenho, tempo de *rendering* e *planning* por quantidade de unidades para a configuração de BVP 1, ou seja, o algoritmo planejador BVP funcionando na íntegra, fica evidente a variação do tempo de *planning* relacionada ao aumento do número de unidades, enquanto os tempos de *rendering* permanecem estáveis abaixo de 10ms. Também fica evidente a queda brusca do valor de FPS mínimo quando os tempos médios de *planning* ultrapassam o limite em torno de 20ms.

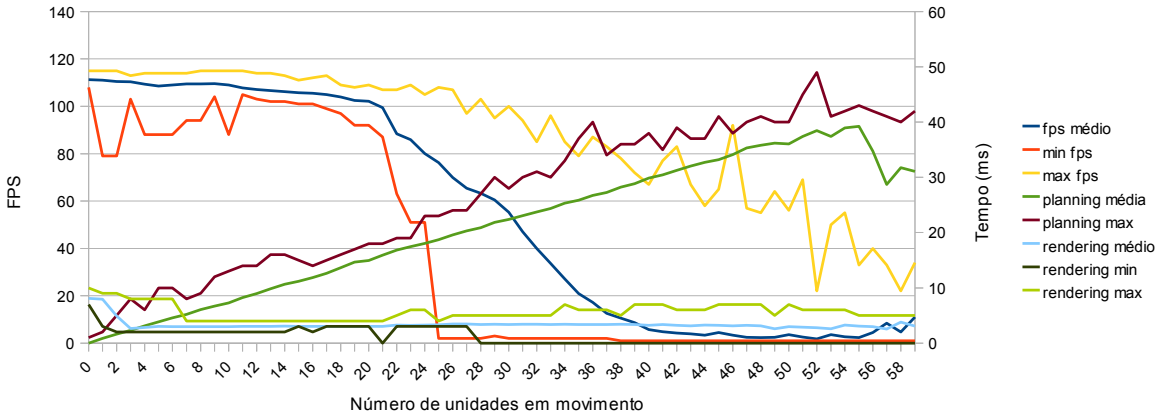


Figura 5.18: FPS, tempo de rendering, e planning para o planejador BVP.

5.4.1.3 Comparação entre planejador BVP e nativo

Para completar a análise de desempenho do planejador BVP rodando no mapa 1, o mesmo é comparado com o planejador nativo em relação ao número de unidades na Figura 5.19. O resultado mais importante mostrado pelo gráfico é o crescente tempo de *planning* do planejador BVP, em relação ao quase estável tempo de *planning* do planejador nativo.

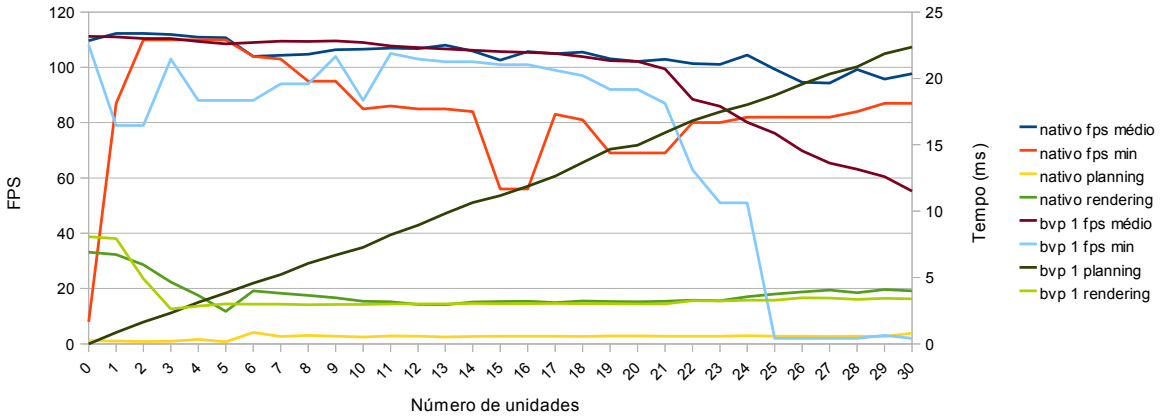


Figura 5.19: Comparação entre o planejador nativo e o planejador BVP.

5.4.2 Resultados dos demais mapas

Os mapas 2 a 5, sendo maiores do que o mapa 1, possibilitaram um maior tempo de simulação, já que os oponentes tiveram mais tempo para evoluir seus exércitos e construir mais unidades antes que ocorressem os encontros e as batalhas entre as unidades inimigas.

O mapa 5 foi especialmente importante nos testes de desempenho, já que é o maior mapa, e também o que possui mais obstáculos e terreno mais acidentado, e por isso, é o mapa que possui mais relevância nos testes de desempenho para mostrar os limites dos algoritmos em condições extremas de funcionamento.

A Figura 5.20 mostra um gráfico contendo as informações de desempenho do planejador nativo rodando no mapa 5, incluindo tempo de *rendering, planning*, e FPS, amostrados conforme o número de unidades. O gráfico mostra o aumento esperado do tempo de *rendering* acompanhando o aumento do número de unidades da simulação. Também apresenta a estabilidade do desempenho médio e do tempo de *planning* mesmo com um aumento significativo do número de unidades, mostrando que o planejador de caminhos nativo escala

bem, pelo menos até a quantidade de 240 unidades se movendo.

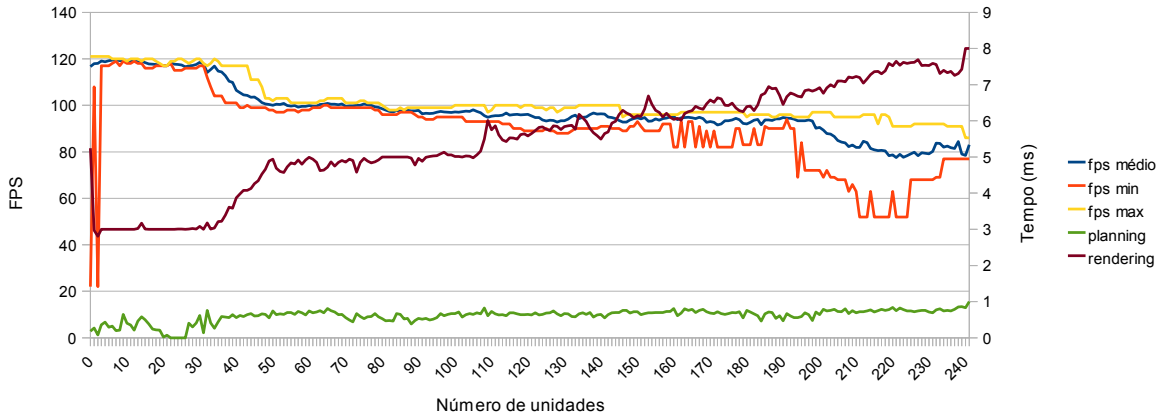


Figura 5.20: FPS, planning e rendering para o planejador nativo.

A Figura 5.21 apresenta o mesmo tipo de gráfico da Figura 5.20, porém contendo os dados referentes ao teste do planejador BVP na configuração 1. Apesar do teste não ter atingido um número tão grande de unidades quanto no teste do planejador nativo, o número de unidades foi alto o suficiente para ir bem além do limite computacional do planejador BVP, e permitir uma clara visualização das condições limite. No gráfico, fica evidente que, para quantidades de unidades maiores do que 40, o FPS cai para níveis inaceitavelmente lentos de interatividade, muito abaixo de 10 *frames* por segundo.

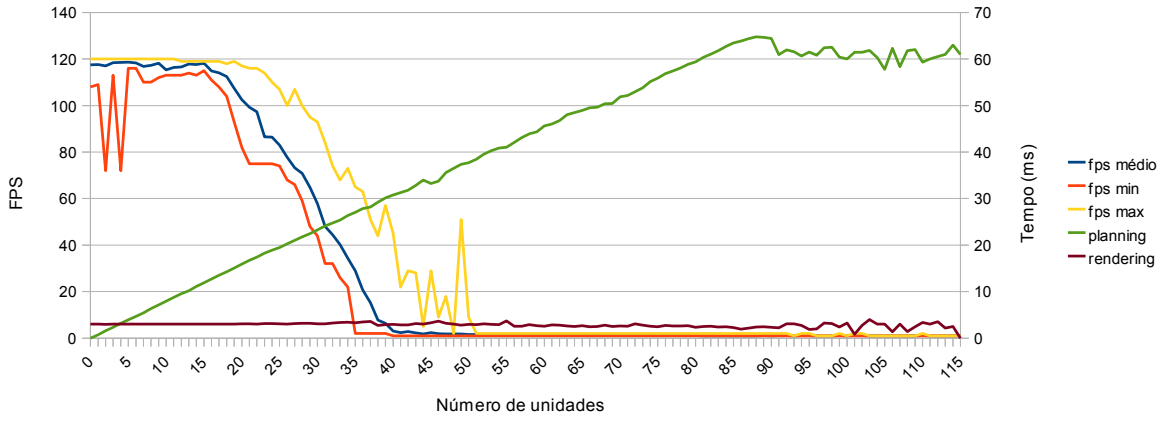


Figura 5.21: FPS, planning e rendering para o planejador BVP 1.

5.4.3 As outras configurações de planejamento

Um problema notado durante a execução dos testes, e que já havia sido percebido durante testes da fase de implementação, foi a ocorrência de unidades presas dentro de fábricas, impedindo a sequência normal da simulação.

O problema ocorre por causa da preferência da inteligência artificial que controla os times não humanos de construir a base muito compacta, com as estruturas muito próximas, e as unidades trabalhadoras em volta das fábricas. Nessa situação, há uma chance muito alta de a unidade recém construída dentro de uma fábrica não possuir um caminho livre para saída da fábrica, pois existem unidades muito próximas do vão de saída. Quando isso acontece, a fábrica fica trancada e não pode ser usada para produzir outras unidades enquanto a unidade não sair de dentro da área interna da fábrica. Se todas as fábricas ficarem trancadas, a

simulação também ficará trancada, já que a inteligência artificial não foi programada para tratar essa situação. É importante salientar que essa situação só ocorre com jogadores artificiais - e também usando o planejador BVP - já que um jogador humano facilmente detecta o trancamento da unidade dentro da fábrica, podendo desobstruir a saída da mesma e retirar a unidade.

A Figura 5.22 mostra um exemplo da ocorrência do problema. A unidade dentro da fábrica, ao centro, não pode sair porque existe uma unidade parada trancando a saída. Em uma situação normal, o planejador de caminhos nativo resolve esta situação facilmente fazendo com que as unidades recém criadas "empurrem" as unidades paradas que estão trancando o caminho. Assim, o tratamento de colisões se encarrega de fazer com que os golpes dos "empurrões" movam as unidades paradas, até que um caminho possível seja aberto para passagem.

Figura 5.22: Unidade trancada dentro da fábrica.

Outro problema decorrente das unidades trancadas dentro da fábrica é que, com a planificação causada pela falta de saída, o vetor de direção não pode ser obtido, e as unidades em questão podem apresentar um movimento oscilatório em direções aleatórias até que o módulo de movimento detecte que a unidade não está andando e cancele o movimento. Os movimentos em direção aleatória podem fazer com que a unidade entre dentro de paredes ou outras unidades, já que a verificação de colisão está desativada na configuração BVP 1. Caso as unidades se movam para dentro de obstáculos, dificilmente poderão sair de lá, posto que nessa situação sempre ocorrerá uma planificação no centro do mapa por presença de obstáculos no centro, e na prática impedindo a unidade de se mover.

A Figura 5.23 mostra um exemplo da situação de unidades entrando dentro dos obstáculos. Na figura são exibidas pelo menos 4 unidades trancadas dentro da base por causa das direções aleatórias causadas por planificações que ocorrem nas situações de falta de caminho livre para sair da fábrica. É importante salientar que, na configuração BVP 2 e demais configurações em que a verificação de colisão está ativada, o problema citado não ocorre, pois as unidades são impedidas de entrarem umas dentro das outras, ou então dentro de obstáculos e fábricas.

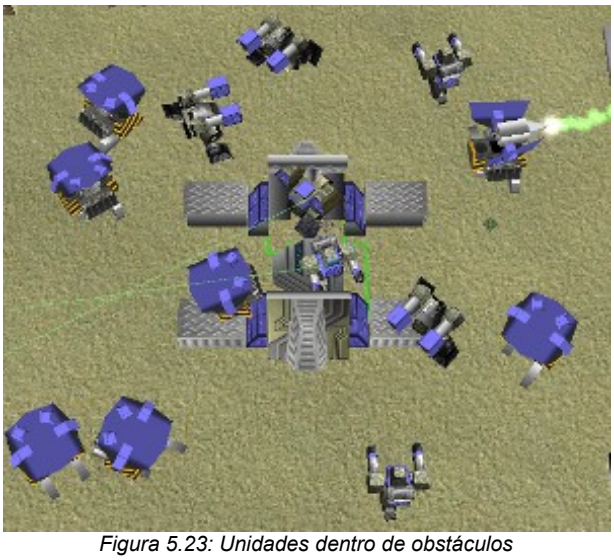


Figura 5.23: Unidades dentro de obstáculos

A Tabela 5.5 mostra a frequência da ocorrência dos travamentos da simulação inteira nos testes de desempenho efetuados, lembrando que cada configuração de teste foi executada 3 vezes:

Tabela 5.5: Travamento da simulação por unidade bloqueada.

Configuração	*Mapa 1*	*Mapa 2*	*Mapa 3*	*Mapa 4*	*Mapa 5*
nativo					
bvp 1	1 de 3	2 de 3		2 de 3	2 de 3
bvp 2					
bvp 3					
bvp 4					

Assim, fica evidente que o planejador BVP sofre uma desvantagem injusta de jogo na *Spring* quando as unidades são controladas pela inteligência artificial, devido principalmente à característica de jogo da inteligência artificial. A característica do terreno não é muito relevante para a ocorrência do problema, e sim a disposição das unidades e estruturas na base feitas pela IA, apesar de que, com bases construídas em terrenos mais acidentados, a chance do problema ocorrer também aumenta, já que existem menos células passáveis na base por causa do terreno acidentado.

Mesmo que o problema do trancamento não seja tão importante quando as unidades são controladas por um jogador humano, a configuração BVP 2 foi especialmente ajustada para tratar a situação, seguindo a mesma lógica usada pelo planejador nativo. Assim, a configuração BVP 2 possui o tratamento de colisões ativado, além de tratar unidades paradas como células livres no mapa local do agente. Dessa forma, quando a unidade é criada dentro de uma fábrica com a saída trancada por unidades trabalhadoras paradas, estas são tratadas como caminho livre, e a unidade recém criada avança para cima das unidades paradas, empurrando-as para fora do caminho através do tratamento de colisão.

O gráfico apresentado na Figura 5.24 mostra uma comparação entre o aumento do número de unidades em movimento no planejador nativo, na configuração BVP 1 em uma situação em

que a mesma trancou por bloqueio das fábricas, e uma configuração BVP 2 funcionando normalmente. Pelo gráfico pode-se observar que, enquanto o planejador nativo e a configuração BVP 2 permitiram um aumento significativo do número de unidades em movimento, na configuração de BVP 1 a quantidade de unidades ficou estagnada devido ao bloqueio das fábricas. Eventualmente as unidades conseguiram sair da fábrica, ou outras fábricas foram construídas, causando a variação apresentada no gráfico.

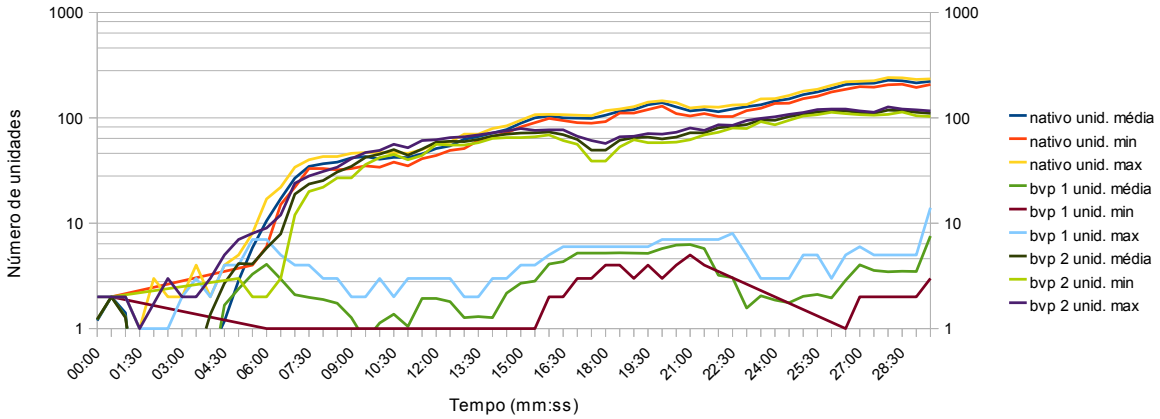


Figura 5.24: Número de unidades no planejador BVP 1 em situação de bloqueio.

A configuração 3 do planejador BVP possui a mesma configuração da versão 2, com exceção da forma de trabalho do mapa global. Na configuração 2, o mapa global gera o vetor gradiente global à partir da informação de planejamento fornecido pelo planejador nativo. Na configuração 3, o mapa global gera o vetor gradiente global simplesmente à partir da direção atual da unidade em relação ao objetivo.

O objetivo do teste da configuração 3 foi verificar se a presença de planejamento nativo no mapa global da configuração BVP afeta de forma significativa os tempos de simulação. Como a configuração 3 não utiliza o planejador nativo para a geração do vetor de direção do mapa global, uma comparação com o resultado da configuração 2 permitiu a verificação do nível de influência dos tempos do planejador nativo quando em uso pelo mapa global.

A comparação entre os resultados dos testes da configuração BVP 2 e 3 podem ser vistos na Figura 5.25. O gráfico apresentando na figura mostra os resultados muito parecidos para os dois tipos de configuração. Pode-se notar também que o tempo de *planning* do planejador nativo durante a simulação usando a configuração BVP 2 é muito pequeno, praticamente irrelevante em relação ao tempo de *planning* do planejador BVP, de onde pode se concluir que a utilização do mapa global usando o planejador nativo tem baixa influência nos resultados dos testes de desempenho do planejador BVP.

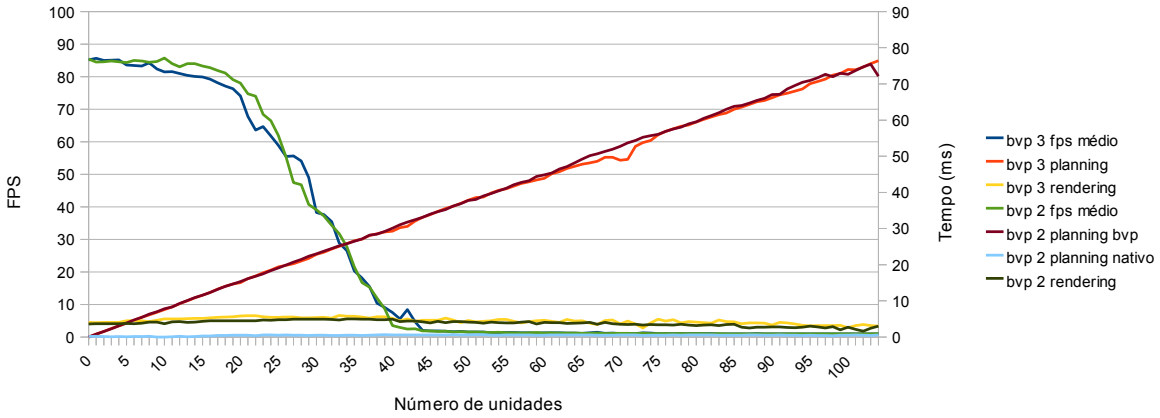


Figura 5.25: Planejador BVP configuração 2 e 3.

Cabe salientar ainda que a utilização do mapa global conforme a configuração 3 representa uma grande desvantagem para o jogador durante o jogo, já que o mapa global simplificado dessa maneira sempre direcionará as unidades direto para o objetivo, ignorando obstáculos grandes e terrenos com custo alto, onde somente a detecção e desvio de obstáculos locais estará atuando através do mapa local da unidade. Nessa situação, as unidades podem até ficar circulando presas atrás de obstáculos grandes ou côncavos quando forem maiores do que o mapa local, impossibilitando a unidade de vencer o obstáculo, contudo sem gerar planificação. A Figura 5.26 mostra a ocorrência de uma unidade bloqueada por um obstáculo côncavo grande: sem a ajuda do mapa global, a unidade é incapaz de contornar o obstáculo.

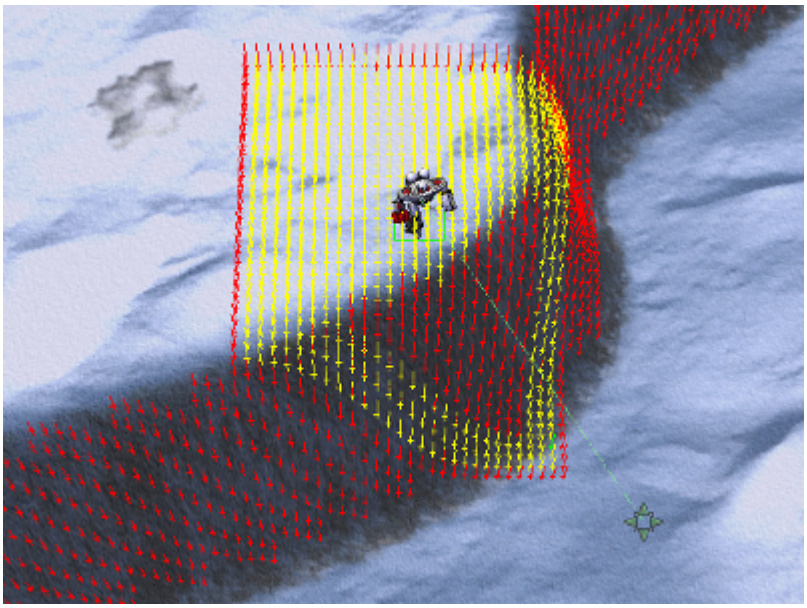


Figura 5.26: Unidade bloqueada por obstáculo côncavo.

Quanto à configuração número 4, teve por objetivo testar o módulo de direção nativo da Spring, usando o controle de colisão local do planejador BVP apenas para o desvio de obstáculos. O controle de direção BVP ajusta a direção da unidade a cada *frame* da simulação, enquanto o controle de direção nativo ajusta a direção da unidade em um *frame* e então anda na direção informada durante vários *frames* da simulação, dependendo da velocidade da unidade. Essa estratégia segue a lógica de que a unidade, igual a um humano, possui uma velocidade limitada de percepção de obstáculos e direção, além de diminuir o número de

chamadas do módulo de movimento ao módulo de planejamento.

A Figura 5.27 mostra um gráfico de comparação de desempenho entre a configuração BVP 1 e 4. No gráfico, é possível observar que o funcionamento das duas configurações é muito semelhante dentro da região em que o desempenho da simulação se encontrava ainda interativo. Alguns testes rápidos durante o jogo também apresentaram resultados indiferenciáveis entre as duas configurações no que diz respeito ao movimento das unidades, levando-se à conclusão de que as duas configurações, da forma como estão propostas, são na prática equivalentes.

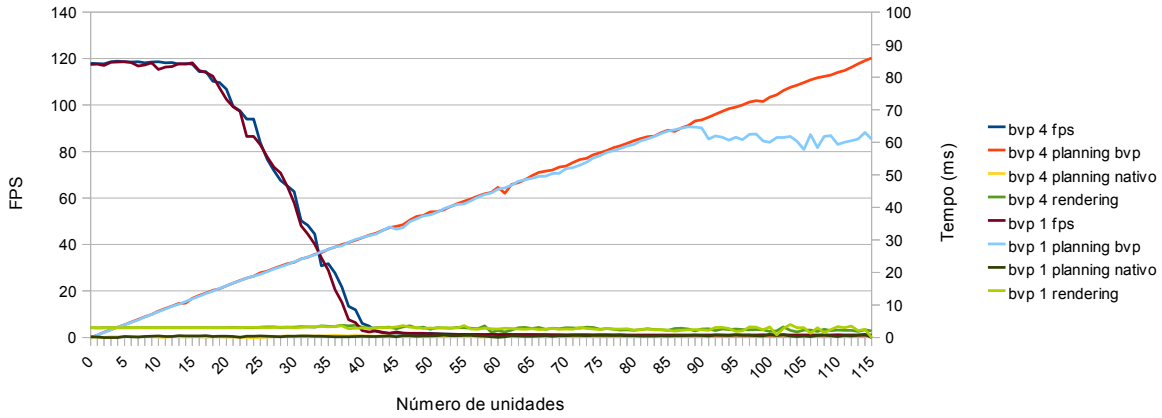


Figura 5.27: Controle de direção nativo e BVP.

5.5 Qualidade

Os testes de qualidade tiveram como objetivo possibilitar uma apreciação visual do caminho gerado pelo planejador BVP, bem como realizar uma comparação do mesmo com o caminho gerado pelo planejador nativo em situações semelhantes. Os testes foram realizados usando-se a mesma configuração dos testes de desempenho.

A Figura 5.28 mostra, desenhados no mesmo mapa, os caminhos gerados pela unidade para chegar do ponto rotulado como "Início" até o ponto rotulado como "Fim", usando tanto o planejador nativo, como o planejador BVP.

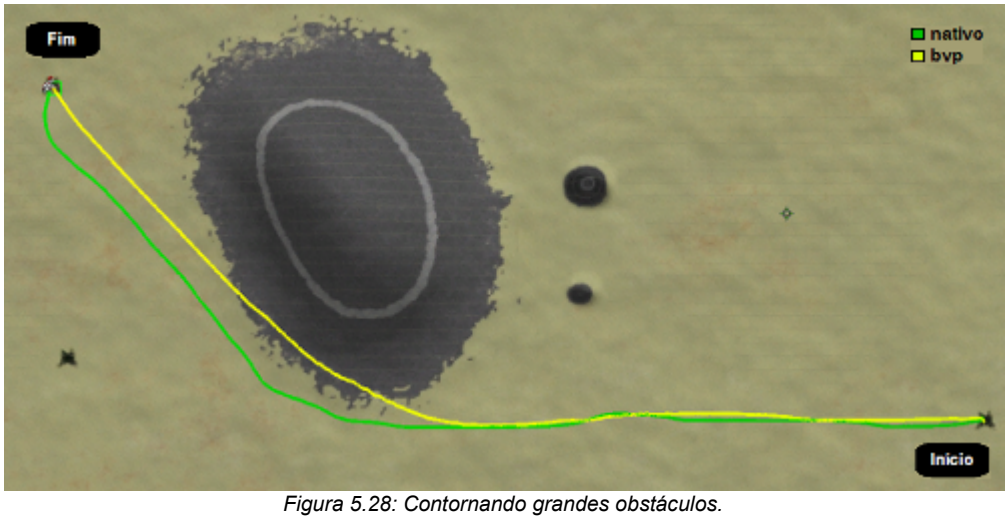


Figura 5.28: Contornando grandes obstáculos.

O caminho feito usando o planejador BVP parece o caminho mais óbvio a ser feito,

enquanto que o caminho nativo possui alguns segmentos diferentes do esperado e pouco artificiais. Além disso, o caminho fornecido pelo planejador BVP tende a ser mais suave do que o caminho gerado pelo planejador nativo, parecendo mais com o caminho natural seguido por uma pessoa do que o caminho gerado pelo planejador nativo.

A Figura 5.29 apresenta os caminhos feitos contornando um obstáculo grande e também passando por vários pontos antes de chegar ao objetivo. Na figura fica mais evidente a tendência dos algoritmos de busca baseados em A* de gerarem caminhos com segmentos estranhos e artificiais: o caminho gerado pelo planejador nativo, perto do início e perto do fim, parece conter alguns efeitos oscilatórios, possivelmente causado pela discretização do mapa em células usado pela função heurística de busca do algoritmo baseado em A*. Lembrando que o algoritmo BVP utiliza o mapa global baseado no planejador nativo, pode ser visto na figura que essa condição não obrigou o planejador BVP a seguir o mesmo caminho do planejador nativo, na verdade suavizando os efeitos oscilatórios do planejador nativo, porém aproveitando o caminho gerado baseado em custo de terreno fornecido pelo planejador nativo, pois de outra forma a unidade teria subido pela montanha, ignorando o alto custo da borda inclinada da montanha.

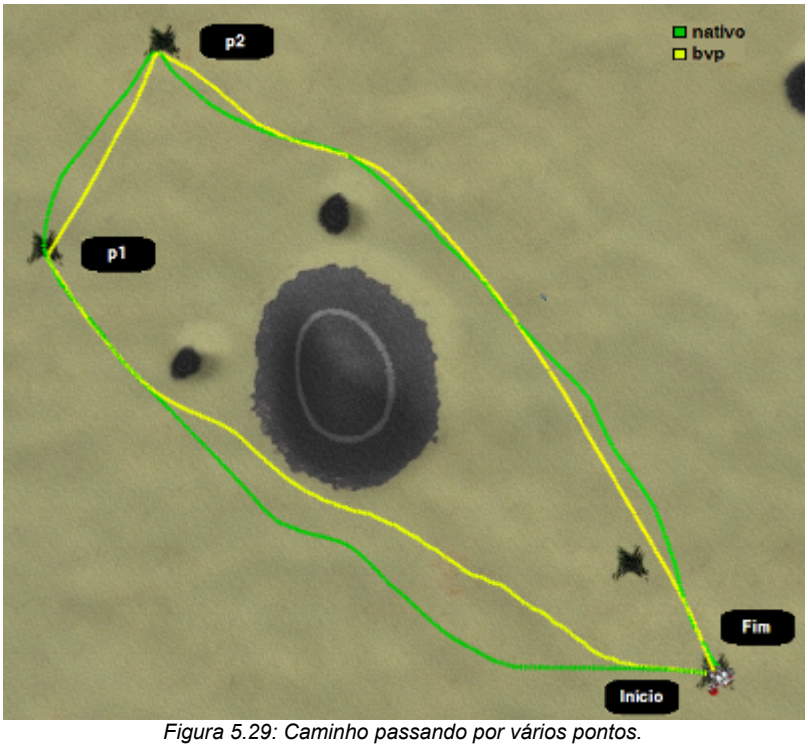
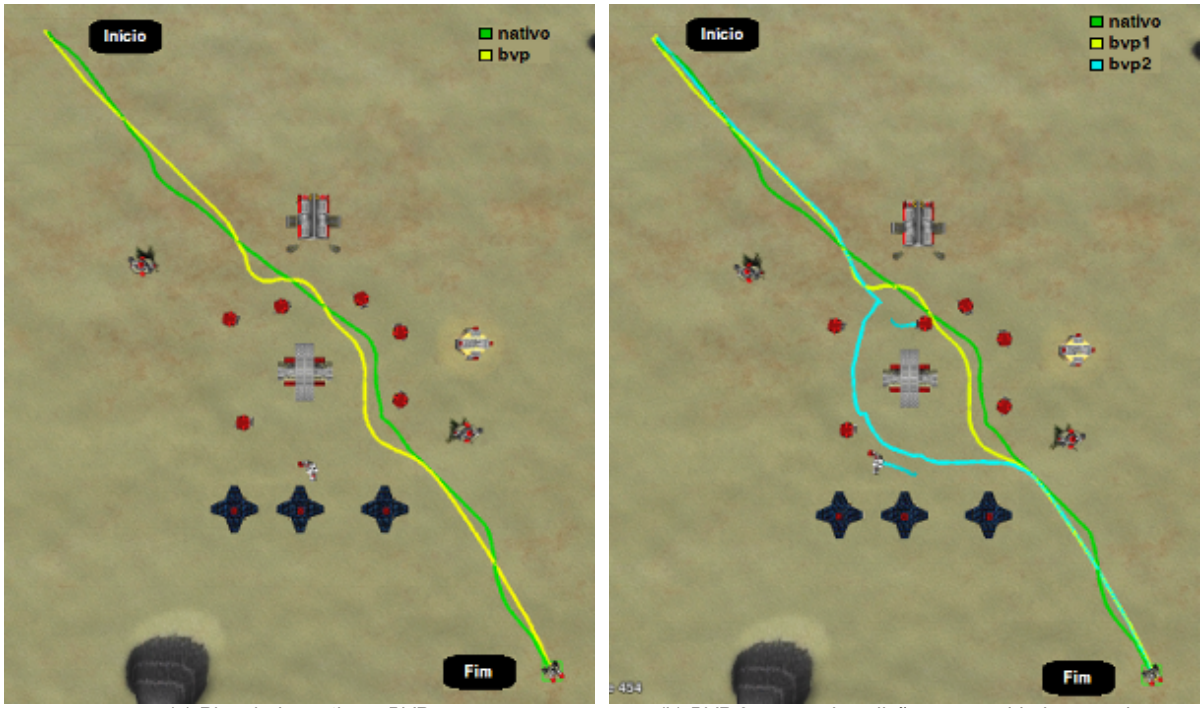


Figura 5.29: Caminho passando por vários pontos.

A Figura 5.30a mostra uma situação de travessia pelo meio da base, contendo vários obstáculos na forma de outras unidades e estruturas. Os caminhos exibidos mostram o caminho suave gerado pelo planejador BVP, juntamente com o caminho gerado pelo planejador nativo, que mais uma vez mostra alguns artefatos artificiais, apesar de cumprir o objetivo de atravessar a base desviando dos obstáculos.

Durante a seção de testes de desempenho foi comentada a fraqueza da configuração 1 do planejador BVP de vencer obstáculos muito aglomerados ou na saída da fábrica, e para contornar o problema foi proposta a configuração 2 do planejador BVP, onde as unidades paradas são tratadas como caminho livre, e a verificação de colisão ativada trata das situações em que as unidades empurram outras unidades para abrir caminho.

A Figura 5.30b mostra a mesma situação da Figura 5.30a, usando-se a configuração 2 do planejador BVP. Na figura pode ser visto que, enquanto o planejador nativo e o planejador BVP 1 atravessaram a base com sucesso sem causar colisões, a configuração 2 do planejador BVP tratou as unidades paradas como caminho livre, e apesar de ter atravessado a base com sucesso, causou algumas colisões, como pode ser visto pelo caminho das duas unidades que se moveram como reação à colisão.



(a) Planejador nativo e BVP. *(b) BVP 2 causando colisões com unidades paradas*

Figura 5.30: Caminho feito pela unidade atravessando a base.

Seguindo a ideia da Figura 5.30, a Figura 5.31 visa mostrar a situação de saída da base pela unidade.

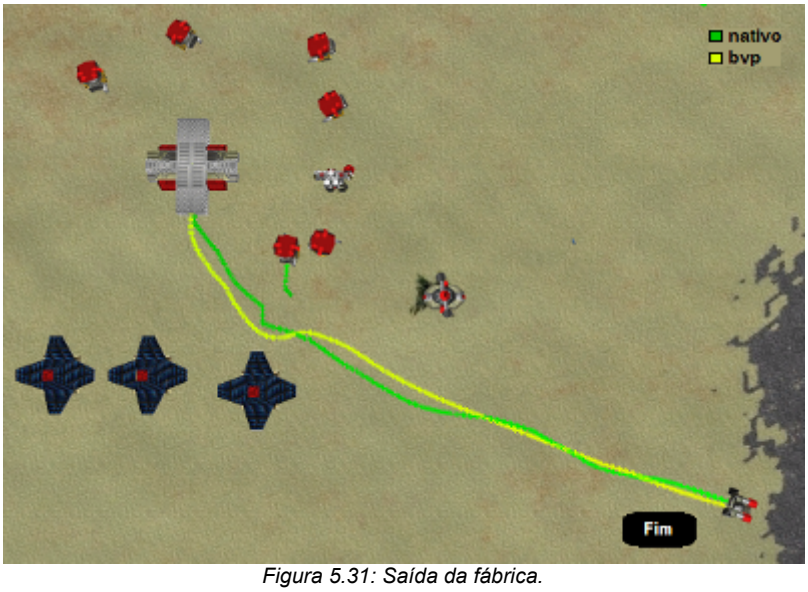


Figura 5.31: Saída da fábrica.

O teste documentado na figura também mostra que o planejador nativo não está livre de colisões com unidades paradas, como pode ser visto pela pequena unidade trabalhadora que se moveu[20] como resultado de um abalroamento leve lateral causado pela unidade controlada pelo planejador nativo.

Para finalizar o teste de saída de fábrica, a Figura 5.32 documenta o caminho feito por uma unidade controlada pelo planejador BVP 2. Enquanto que o planejador nativo tocou de leve lateralmente a unidade na passagem, a unidade em movimento controlada pelo planejador BVP 2 causou uma colisão frontal com a unidade trabalhadora parada, provocando o seu deslocamento.

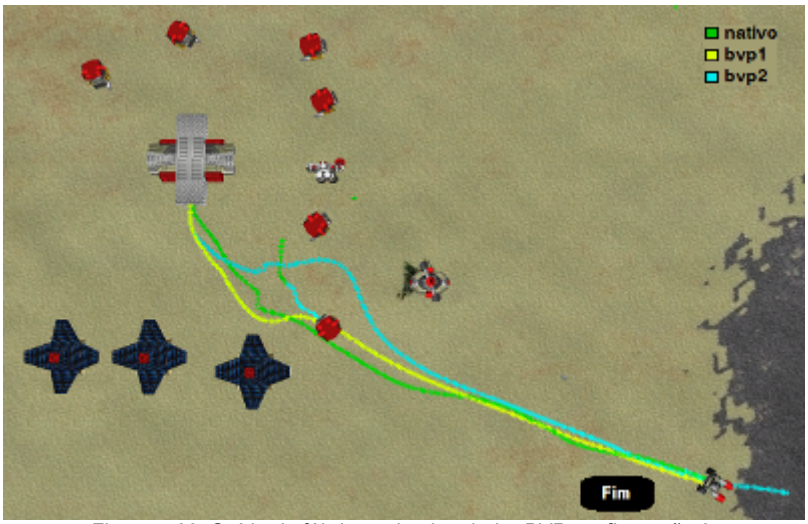
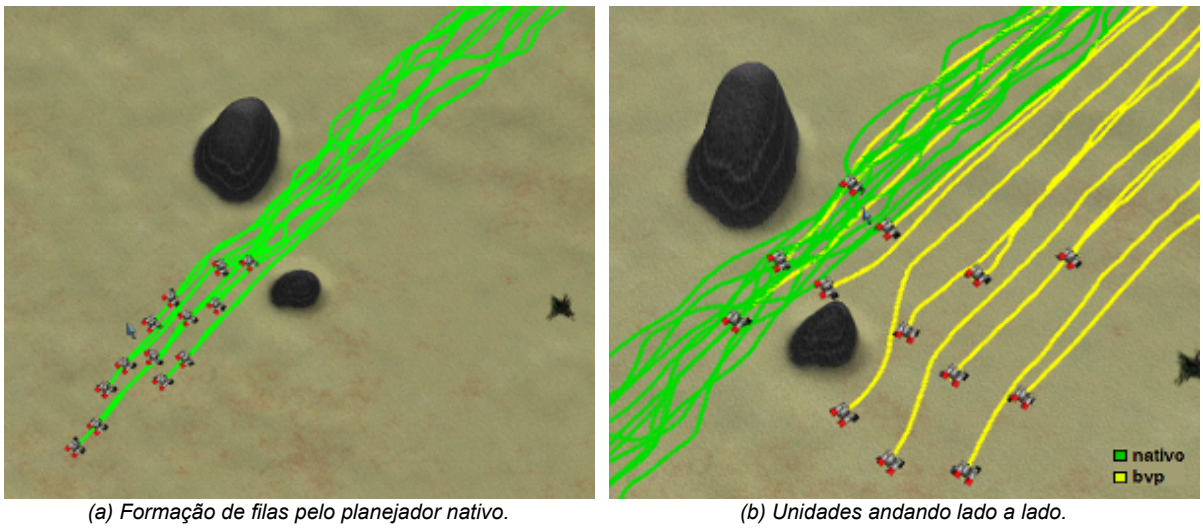


Figura 5.32: Saída de fábrica pelo planejador BVP configuração 2.

Na Figura 5.33 o teste foi realizado com a criação de um grupo de 13 unidades, e a verificação do caminho percorrido pelo grupo até um objetivo comum e distante. A Figura 5.33a mostra a tendência do planejador nativo de formar uma fila de unidades, já que todas procuram seguir o mesmo caminho gerado pelo planejador baseado no algoritmo A*. A *Spring* emprega o recurso de "mapa de calor[21]" pra tentar evitar que as unidades sigam o mesmo caminho, contudo o resultado obtido ainda tende a ser uma fila de unidades seguindo na mesma direção. A Figura 5.33b apresenta a mesma situação do grupo de 13 unidades, porém controladas pelo planejador BVP. Como pode ser visto, as unidades tendem a andar lado a lado, evitando andar em fila, já que o gradiente descendente atrás de um obstáculo formado pela unidade à frente automaticamente direciona a unidade para o lado, saindo da traseira da unidade à frente.

[20] Apesar da figura não mostrar, o teste com o planejador BVP foi executado com a unidade trabalhadora posicionada no mesmo local do que no teste com o planejador nativo.
[21] Mais informações na Seção 4.5, página 36.



(a) Formação de filas pelo planejador nativo. *(b) Unidades andando lado a lado.*

Figura 5.33: Unidades se movendo em grupo.

A Figura 5.34 mostra a chegada das unidades controladas pelo planejador nativo no objetivo. Como as unidades tendem a seguir em fila, também tendem a chegar em fila no objetivo. A situação pode representar uma desvantagem estratégica, na medida em que a primeira unidade da fila poderá chegar sozinha em um local contendo muitos inimigos, e começará a ser atacada sozinha até que os colegas de trás da fila também cheguem na área.

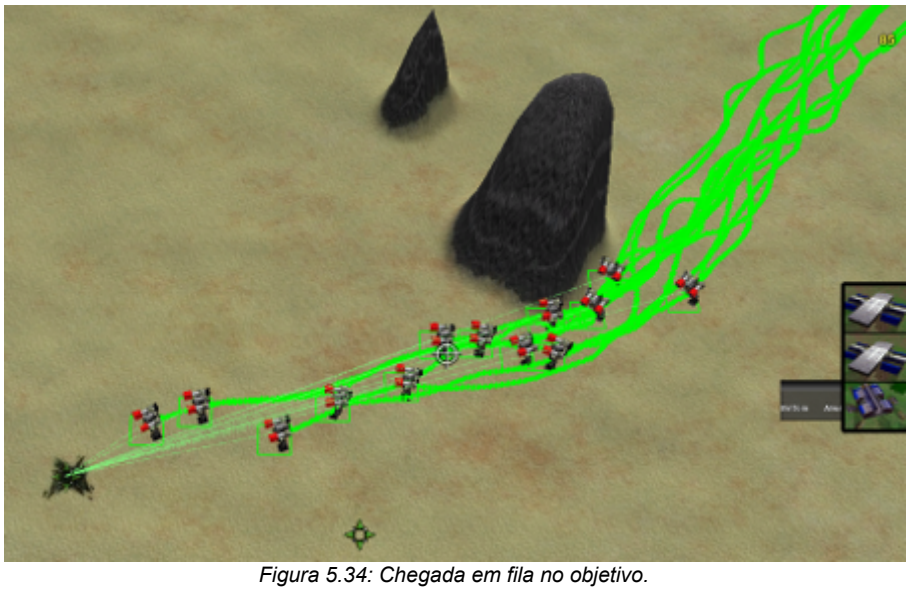


Figura 5.34: Chegada em fila no objetivo.

A Figura 5.35 mostra a chegada de unidades controladas pelo planejador BVP no objetivo. Diferente do planejador nativo, com o planejador BVP as primeira unidades chegaram no objetivo lado a lado com as companheiras, devido à característica inerente do planejador BVP de as unidades evitarem se movimentar em fila.

No jogo de estratégia baseado em combate entre unidades, a chegada das unidades ao objetivo em formação lado a lado pode representar um fator de vantagem estratégica, já que várias unidades chegarão ao objetivo ao mesmo tempo, começando a atacar ao mesmo tempo, enquanto que na chegada em fila, enquanto uma unidade já chegou e está sendo atacada, as outras ainda estão chegando e se posicionando para o combate.

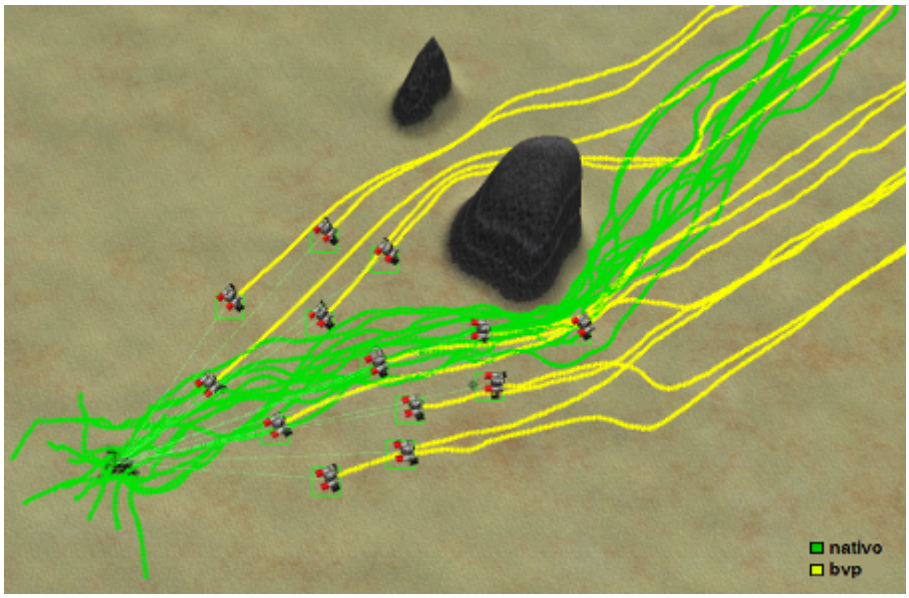


Figura 5.35: Chegada lado a lado no objetivo.

Para completar a análise do caminho percorrido pelo grupo de unidades, a Figura 5.36 mostra o caminho completo percorrido pelas 13 unidades controladas pelo planejador nativo, e a Figura 5.37 mostra a comparação entre os caminhos gerados pelos dois planejadores.

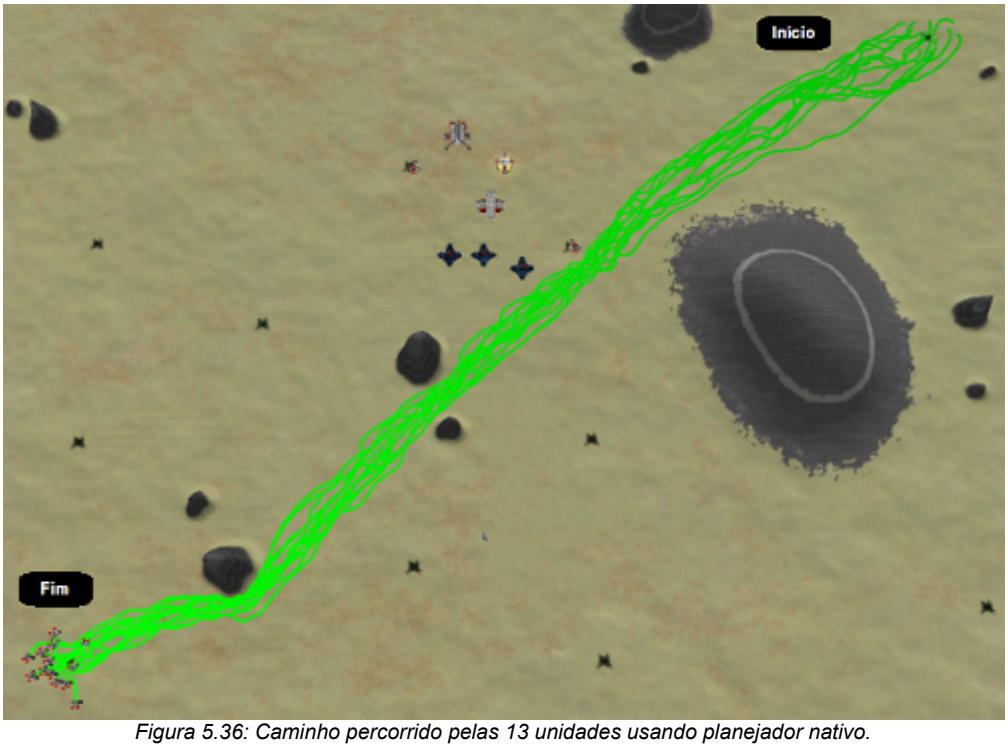


Figura 5.36: Caminho percorrido pelas 13 unidades usando planejador nativo.

Como foi visto nos outros testes, na Figura 5.37 os caminhos das unidades controladas pelo planejador BVP tendem a guiar as unidades seguindo em paralelo e lado a lado, diferente do planejador nativo, que por sua natureza, tende a agrupar as unidades em fila.

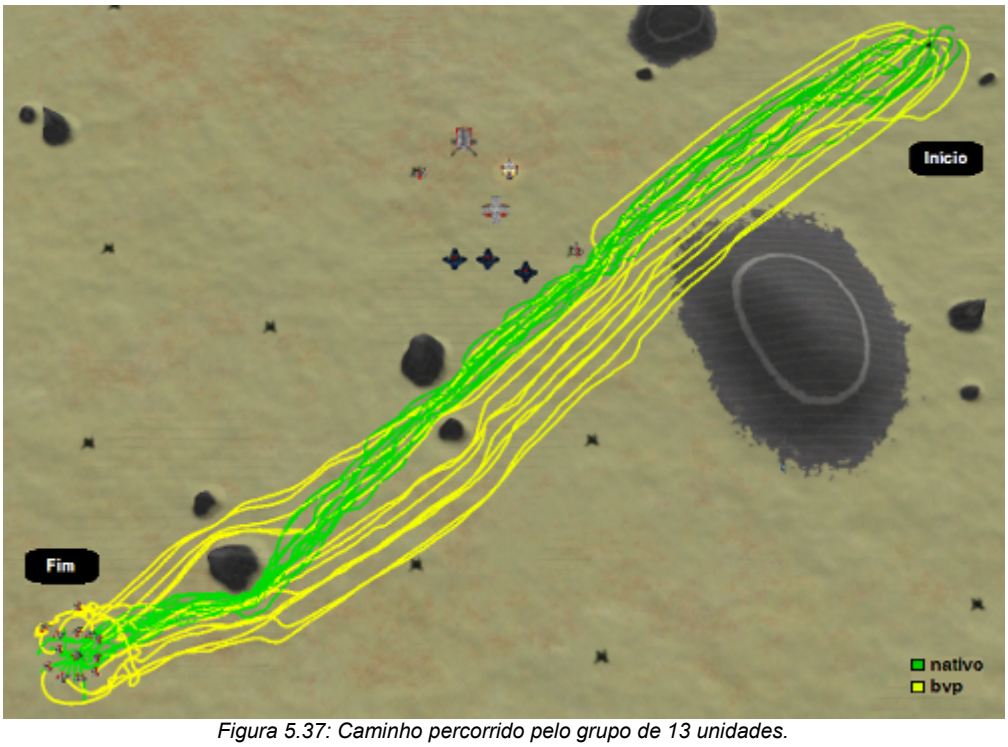


Figura 5.37: Caminho percorrido pelo grupo de 13 unidades.

5.6 Vantagem estratégica

Testes de vantagem estratégica foram feitos através da configuração de jogos em que um dos oponentes utiliza o planejador nativo, enquanto o outro utiliza o planejador BVP. Os resultados dos testes foram anotados e comparados em busca da ocorrência de uma quantidade maior de vitórias para algum dos oponentes, baseada em uma possível vantagem estratégica fornecida pelos caminhos gerados pelos dois tipos de planejador de caminhos.

Os testes foram executados com as mesmas configurações apresentadas nos testes de desempenho. O mapa utilizado foi o mapa 3, por ser uma mapa relativamente simétrico nas posições iniciais dos jogadores, por ser bem plano, e permitir que os jogos geralmente acabem dentro do limite de tempo de 30 minutos, apesar de também possibilitar disputas com um número elevado de unidades. Para as partidas que ultrapassaram o tempo limite, foi computada vitória para o time que possuísse mais unidades em jogo, e em caso de empate novamente, o fator de desempate foi o número de unidades produzidas, o número de unidades destruídas do oponente, e o número de unidades destruídas pelo oponente.

No total, foram executados 35 testes, que rodaram durante 12 horas, e geraram resultados que podem ser visualizados na Tabela 5.6.

Tabela 5.6: Resultados dos testes de estratégia.

Configuração	*Vitória no limite de tempo*		*Tempo esgotado*		*Total de vitórias*		
	nativo	*bvp*	*nativo*	*bvp*	*nativo*	*bvp*	*Partidas*
nativo x bvp 1	13	4	1	0	14	4	18
nativo x bvp 2	8	8	1	0	9	8	17

Através da tabela, pode-se verificar que a configuração BVP 1 possui uma desvantagem

forte em relação ao planejador de caminhos nativo. Na verdade a ocorrência de unidades trancadas dentro das fábricas é o fator chave relevante que desfavorece o uso da configuração BVP 1 controlada pela inteligência artificial. Durante a ocorrência de unidades trancadas, a produção de unidades do jogador parou, enquanto a produção de unidades do oponente continuou funcionando, resultando em um exército maior do oponente disponível durante o combate.

Quanto à configuração BVP 2, apresentou praticamente um equilíbrio entre o número de vitórias e derrotas em relação ao oponente controlado pelo planejador nativo. O fato da configuração BVP 2 não possuir o problema de unidades trancadas dentro da base possibilitou que a inteligência artificial disputasse em condições de igual para igual contra o jogador controlado pelo planejador nativo.

Em relação à presença de vantagens estratégicas na utilização do planejador BVP, o teste mostra que a configuração 1 possui uma desvantagem no uso com jogador controlado por IA, contudo, a ocorrência de unidades bloqueadas nas fábricas pode facilmente ser resolvida por um jogador humano. Além disso, a tendência em construir bases compactas e estruturas muito próximas é uma característica da inteligência artificial usada, e nada garante que outros módulos de IA, ou até mesmo jogadores humanos, escolham essa estratégia de construção.

Já a configuração 2 se mostrou neutra na presença ou ausência de vantagens ou desvantagens, mas de qualquer forma, outros módulos de IA e também os jogadores humanos poderiam explorar a seu favor o fato das unidades em grupo evitarem andar em filas e andarem lado a lado, como foi mostrado na Seção 5.5.

5.7 Modificações, otimizações e outros testes

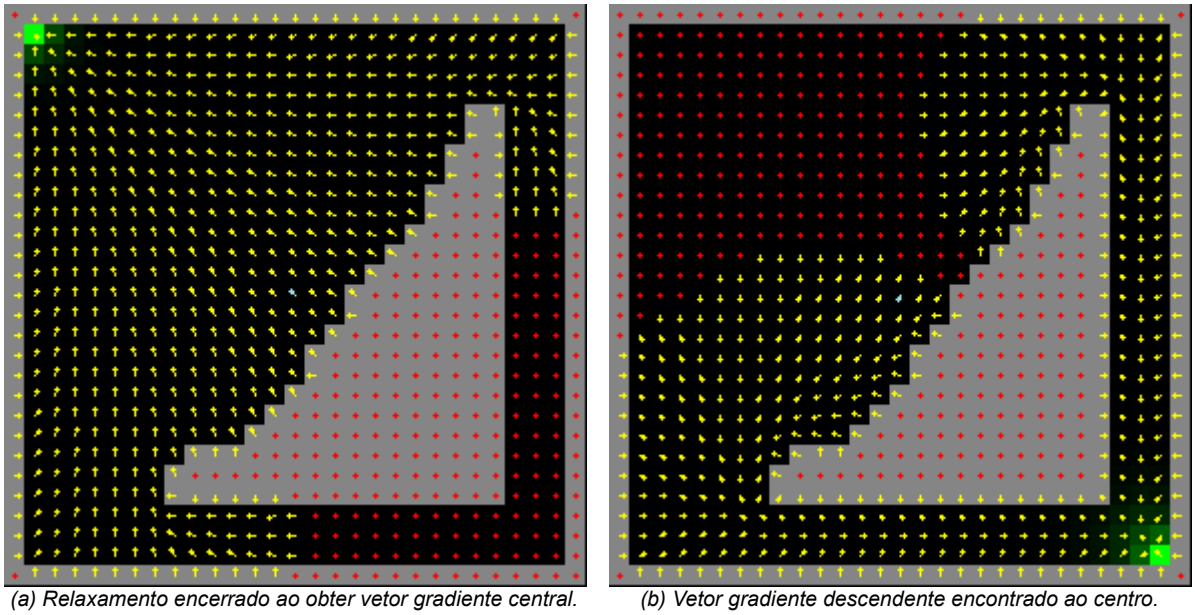
5.7.1 Convergência do processo de relaxamento

O método de convergência adotado no processo de relaxamento do planejador BVP envolve a aplicação do método de relaxamento de Gauss-Seidel por um determinado número de iterações, até que um limite de iterações ou de erro acumulado ultrapassem um determinado patamar. Durante os testes, foi usado um valor de limite de iterações de 100 relaxamentos, e um valor de limite de erro acumulado de zero, bem como foi mostrado que os valores utilizados resultavam, na maioria dos casos, em campos potenciais livres de planificação no centro do mapa - uma condição necessária para que o agente possa usar o mapa local para se mover.

Contudo, alguns testes também mostraram que campos potenciais mais simples ou com poucos obstáculos convergem rapidamente, e já apresentam um centro do mapa livre de planificação em poucas iterações de relaxamento, causando um possível desperdício de cálculo computacional realizando relaxamentos extras. Além disso, testes usando o limite de erro acumulado mostraram que a grande variação de ordem de grandeza do limite de erro acumulado em situações variadas de configuração de obstáculos no mapa dificulta a escolha de um valor fixo de limite de erro que funcione bem para todas as situações sem desperdiçar muitos ciclos de relaxamento.

Em vista disso, uma proposta inicial para otimização do processo de relaxamento poderia ser o monitoramento por planificação da região central do mapa local, e a finalização do processo de relaxamento quando for possível encontrar um vetor gradiente descendente no

centro do mapa. As Figuras 5.38a e 5.38b mostram dois exemplos de relaxamento encerrados quando foi detectado um vetor gradiente válido no centro do mapa (flecha azul na célula). Considerando uma varredura do processo de Gauss-Seidel seguindo da esquerda para a direita, e de cima para baixo, o campo potencial da Figura 5.38a precisou de apenas 1 relaxamento para ser construído, e o campo potencial da Figura 5.38b precisou de 36 relaxamentos. Como pode ser constatado, estes valores de relaxamento são mais baixos do que o valor limite de relaxamento ajustado para 100, o que mostra um ganho razoável de desempenho evitando os passos extras de relaxamento.



(a) Relaxamento encerrado ao obter vetor gradiente central. *(b) Vetor gradiente descendente encontrado ao centro.*

Figura 5.38: Encerramento adiantado do processo de relaxamento.

Contudo, a simplificação do processo de relaxamento dessa forma, antes da ocorrência da convergência dos valores de potencial, pode não garantir um campo potencial livre de mínimos locais. Entretanto, pela própria definição do método de construção do campo potencial usando a Equação de Laplace de forma a tornar o método livre de mínimos locais, o uso do método de relaxamento de Gauss-Seidel implicará que o valor em cada passo de cada célula é representado por uma série de somatórios das células vizinhas do passo atual, e das células vizinhas das vizinhas no passo anterior, e das células vizinhas das células vizinhas no outro passo anterior, seguindo assim sucessivamente. Como todas as células são inicializadas com o valor 1, e somente o objetivo é inicializado com o valor 0, no momento em que uma das células possuir o valor menor do que 1, é porque na série de células somadas representando os passos anteriores haverá pelo menos uma conexão direta com a célula objetivo, indicando que existe pelo menos um caminho entre a célula em questão e o objetivo. Essa situação também vale para o método quando em uso com o vetor de comportamento, já que na equação de atualização do campo potencial, o vetor de comportamento só possui influência quando existe uma diferença de potencial na vizinhança.

Assim, a otimização proposta consegue garantir que, se for encontrado um vetor gradiente no centro do mapa local, já existe pelo menos um caminho válido entre o centro e o objetivo. Todavia, não há garantia de que o caminho seja o melhor caminho possível, o que não é um problema na implementação proposta, nem de que esteja suavizado. Para garantir uma melhor suavização do caminho, podem ser feitos alguns relaxamentos extras após a detecção de

gradiente no centro do mapa, o que possibilitará uma melhor propagação do potencial do objetivo até a vizinhança da célula central, suavizando um pouco mais as direções de gradiente obtidas.

Considerando a otimização proposta, a direção de varredura do processo de relaxamento também influencia diretamente o número de relaxamentos salvos pela otimização. As Figuras 5.38a e 5.38b mostram um exemplo da ocorrência dessa situação, onde o objetivo intermediário posicionado em um ponto mais próximo do início da direção de varredura propicia uma melhor propagação do potencial do objeto, melhorando a otimização e poupando a execução de mais relaxamentos.

Em vista disso, e para auxiliar ainda mais o campo à convergir mais rapidamente, a direção de varredura deve ser ajustada conforme o quadrante em que estiver posicionado o objetivo intermediário, de forma que a célula objetivo seja varrida o mais cedo possível durante o processo de atualização dos potenciais. Uma maneira fácil de fazer isso é dividindo-se o campo potencial do mapa local em 4 quadrantes, e ajustando-se o início e direção da varredura para varrer primeiramente o quadrante contendo o objetivo intermediário, e dessa forma propagar o potencial da célula objetivo o quanto antes dentro do processo de propagação.

À partir de todas as otimizações propostas, segue o algoritmo de relaxamento otimizado:

```
para cada mapa local de unidade :
   prepara o campo potencial
   relax_extra ← limite extra de relaxamentos para melhor suavização
   contador de relaxamentos ← 0
   erro acumulado ← 0
   enquanto não atingir limite de relaxamento ou de erro acumulado :
      verifica potencial do centro do mapa
      se o potencial do centro < 1.0 então
         decrementa relax_extra
         encerra o laço se relax_extra < 0
      fim se
      erro acumulado ← 0
      para cada célula do campo potencial
         varre as células conforme o quadrante do objetivo intermediário
         atualiza o potencial da célula
         atualiza o erro acumulado
      fim para
      incrementa contador de relaxamentos
   fim enquanto
fim para
```

A Figura 5.39 ilustra um gráfico com os resultados de desempenho da aplicação da otimização proposta implementada dentro do planejador BVP. Para obter os resultados do gráfico, 4 testes foram rodados usando o planejador BVP configuração 2, no mapa 4, usando um valor de relaxamento extra de otimização de 4 relaxamentos à título de melhora na suavização, e demais configurações ajustadas com os mesmos valores dos testes de desempenho da Seção 5.4. Os valores selecionados para exibição no gráfico foram os obtidos

da partida em que mais unidades foram utilizadas em jogo.

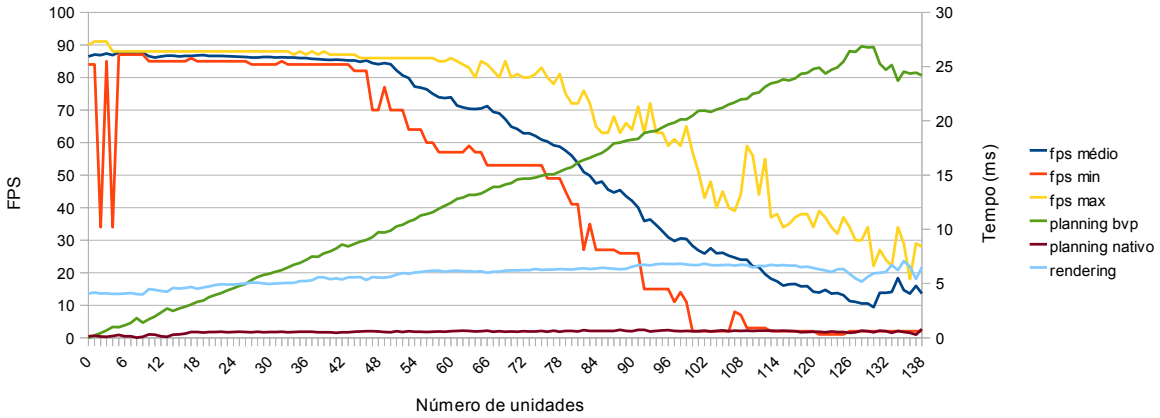


Figura 5.39: FPS, e tempo de planning e rendering usando o relaxamento otimizado.

Se o gráfico for comparado com os patamares limite de FPS e número de unidades apresentados pelo planejador BVP na seção de testes, o mesmo mostra um visível ganho de desempenho na simulação usando o processo de relaxamento otimizado. Durante os testes de desempenho, foi mostrado que o planejador BVP sem otimização no processo de relaxamento começava a provocar uma queda significativa de desempenho abaixo de valores de 20 FPS à partir de 30 à 40 unidades em movimento. Usando-se o processo de relaxamento otimizado, foi possível obter uma taxa de FPS média de 30 *frames*-por-segundo com 100 unidades em movimento, demonstrando um ganho de velocidade de mais de 100%.

A fim de proporcionar uma comparação direta entre o processo original e o processo otimizado, a Figura 5.40 mostra em um mesmo gráfico os resultados da execução do processo original de relaxamento e do processo otimizado, ambos rodados no mesmo mapa de número 4 e com a configuração 2 do planejador BVP. O gráfico mostra que a taxa de crescimento do tempo gasto com *planning* com a configuração não otimizada é mais do que 2 vezes maior do que a taxa de crescimento do tempo de *planning* para a versão usando o processamento de relaxamento otimizado. Dessa forma, é possível atingir um desempenho melhor mesmo com mais unidades se movendo em jogo, permitindo uma melhor percepção de fluidez da simulação durante a partida.

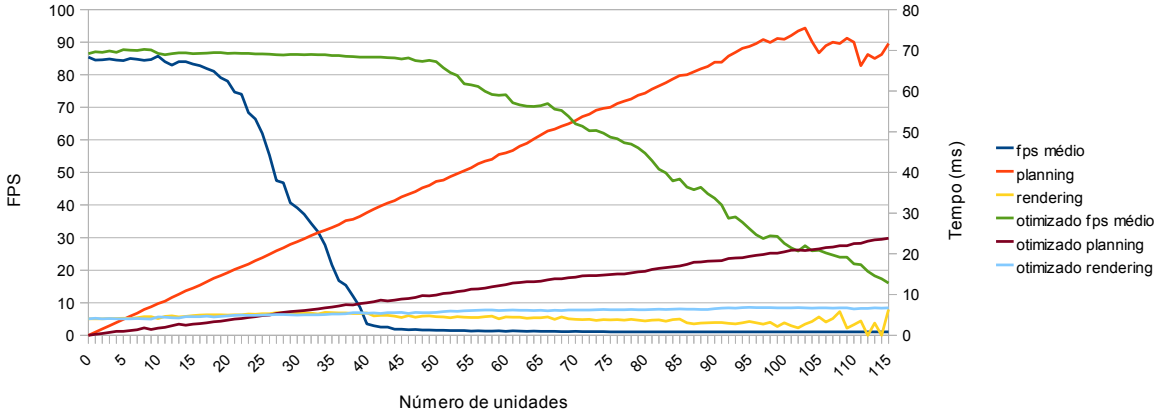


Figura 5.40: Processo de relaxamento original e otimizado.

Para completar a análise do processo de relaxamento otimizado, o gráfico apresentado na Figura 5.41 mostra uma comparação da quantidade de relaxamentos realizados pelo

planejador BVP sem otimização de relaxamento e com otimização de relaxamento. Em vista da otimização no processo de convergência, o número de relaxamentos feitos na simulação com otimização é muito menor do que o número de relaxamentos do processo original.

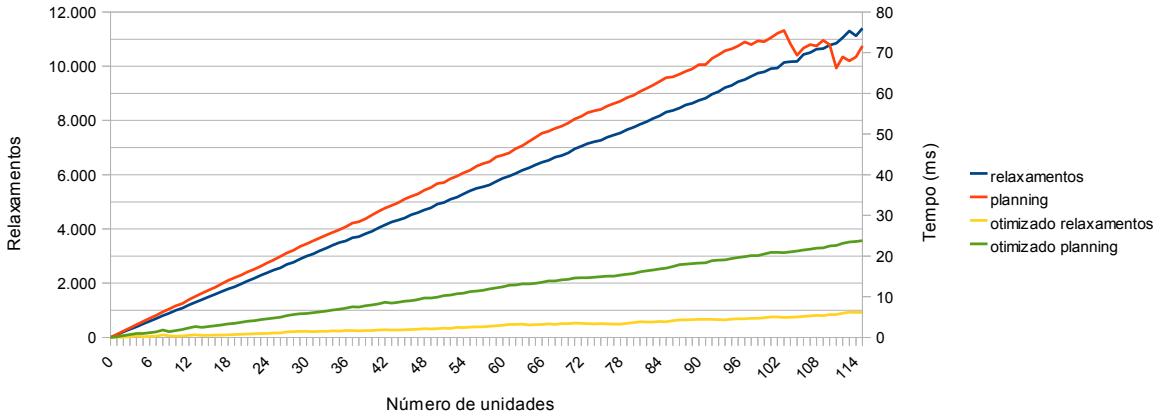


Figura 5.41: Relaxamentos e tempo de planning.

5.7.2 Outros testes

A modularidade e o sistema de *plugins* da Spring permitiu uma fácil mudança de módulo de jogo, mapas diferentes, e diferentes tipos de inteligência artificial.

A Figura 5.42 mostra um exemplo de batalha usando o módulo de jogo *Kernel Panic* versão 3.7 rodando como planejador BVP. O módulo de jogo *Kernel Panic* disponibiliza para a *Spring Engine* um novo modelo de jogo, juntamente com um conjunto de mapas e uma inteligência artificial especialmente construídos sobre uma temática computacional, onde os times representam o "Hacker", a "Rede" e o "Computador", e todas as unidades e demais características de jogo estão interligadas com componentes computacionais derivados da temática de cada time.

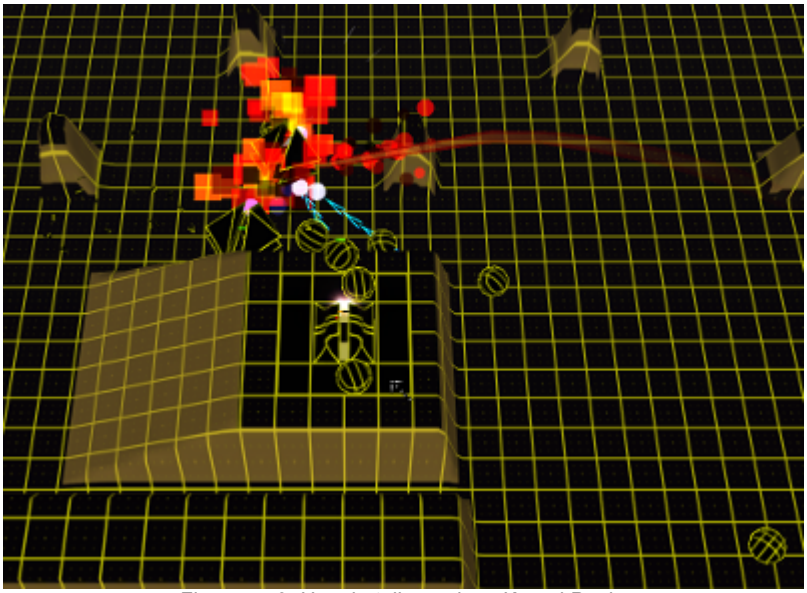


Figura 5.42: Uma batalha no jogo Kernel Panic.

A inteligência artificial que acompanha o jogo *Kernel Panic* funcionou normalmente e atuou conforme o esperado usando o planejador de caminhos BVP. Contudo, alguns dos outros tipos de *plugins* de inteligência artificial também testados utilizam o módulo de

planejamento de caminhos apenas para requisitar a geração de caminhos entre pontos para uso pela própria inteligência artificial, e não para a geração de caminhos para unidades em movimento. Dessa forma, algumas das requisições recebidas pelo planejador de caminhos são originadas do módulo de inteligência artificial, e são executadas sem o fornecimento dos dados da unidade em movimento requisitante, mas apenas os pontos de origem e de chegada do caminho, seguidos por uma requisição de obtenção do vetor de pontos referente ao caminho a ser gerado. O módulo de inteligência artificial então usa o vetor de pontos obtido para controlar a simulação e movimento das unidades de uma maneira bem particular à cada *plugin* testado, e evitando o uso do controle de movimento nativo.

Pela própria natureza do planejamento de caminhos BVP, algumas das chamadas não puderam ser atendidas com sucesso, como a chamada de obtenção de vetor de pontos intermediários representando o caminho total a ser percorrido, utilizada por alguns dos módulos de inteligência artificial, e dessa forma os módulos de inteligência artificial que não estavam preparados para receber uma falha durante a requisição de planejamento levaram a *Spring* à gerar um erro de execução, encerrando o jogo.

5.8 Trabalhos derivados

5.8.1 Código fonte da implementação

O código fonte desenvolvido para a implementação deste trabalho se encontra disponível na Internet sob licença GPL, de forma que outros pesquisadores possam fazer livre uso do código, permitindo facilmente estender este trabalho, realizar outros tipos de teste à respeito de planejamento de caminhos, ou até mesmo usar o código como referência para modificações ou implementações do planejador de caminhos proposto ou ainda modificações e estudos da própria *Spring Engine*.

O código fonte pode ser obtido através de uma visita ao seguinte sítio da Internet:

`http://code.google.com/p/springenginebvppathplanning/`

O código fonte também pode ser obtido através de *checkout* anônimo usando o programa de versionamento SVN no seguinte endereço:

`http://springenginebvppathplanning.googlecode.com/svn/trunk/spring/rts`

Para compilar o código fonte conforme a modificação e implementação executada neste trabalho, fica como recomendação de uma maneira simples e rápida que o leitor obtenha a versão mais recente do código fonte da *Spring Engine* disponível em (SPR 2010), compile e monte com sucesso um executável funcional da *Spring Engine* no seu ambiente de desenvolvimento preferido e no seu sistema operacional preferido, conforme instruções constantes na página de suporte ao desenvolvimento da *Spring*, e depois disso substitua os arquivos existentes na pasta "rts" da estrutura de código fonte da *Spring* pelo conteúdo da pasta "rts" modificada por este trabalho e disponível no endereço informado anteriormente.

Além da implementação do planejador de caminhos BVP para a *Spring Engine*, também se encontram disponíveis no sítio de Internet informado dois programas desenvolvidos a fim de auxiliar na execução dos testes deste trabalho.

O primeiro programa auxiliar desenvolvido foi um editor de mapas e campos potenciais. O programa possui uma interface gráfica que permite ao usuário criar e editar mapas e campos

potenciais, definindo os pontos livres, bloqueados e de objetivo, visualizando na tela os vetores gradiente calculados para cada célula do mapa seguindo a proposta do planejador BVP. A Figura 5.43 mostra um exemplo da tela do programa.

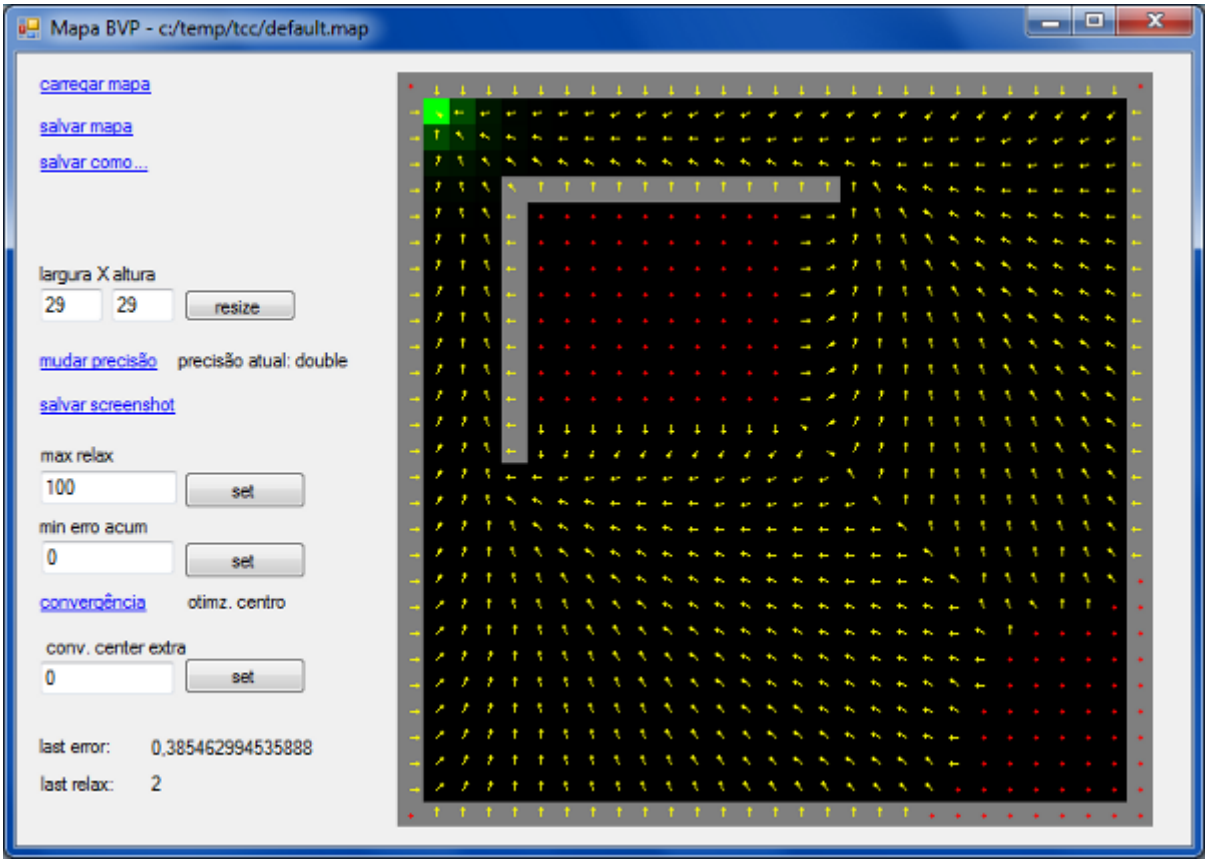


Figura 5.43: Programa editor de mapas e campo potencial.

Os pontos passáveis do mapa pode ser desenhados diretamente com o uso do botão esquerdo do *mouse*, enquanto que o ponto de objetivo pode ser marcado através do uso do botão direito do *mouse*. Além disso, as principais configurações de cálculo do campo potencial podem ser controladas pelo editor, possibilitando uma rápida visualização dos efeitos causados pelas modificações em tempo real. O programa permite ainda que sejam geradas capturas da imagem do campo potencial para arquivos do tipo PNG ou JPEG, recurso que foi usado para construir alguns dos exemplos contidos neste trabalho. Além da captura de imagem, os próprios mapas podem ser armazenados e carregados de arquivos XML contendo internamente os dados serializados dos mapas.

O segundo programa auxiliar desenvolvido para uso neste trabalho trata-se de uma interface para centralização da execução das baterias de testes executados na *Spring*. A Figura 5.44 mostra um exemplo da interface gráfica do programa.

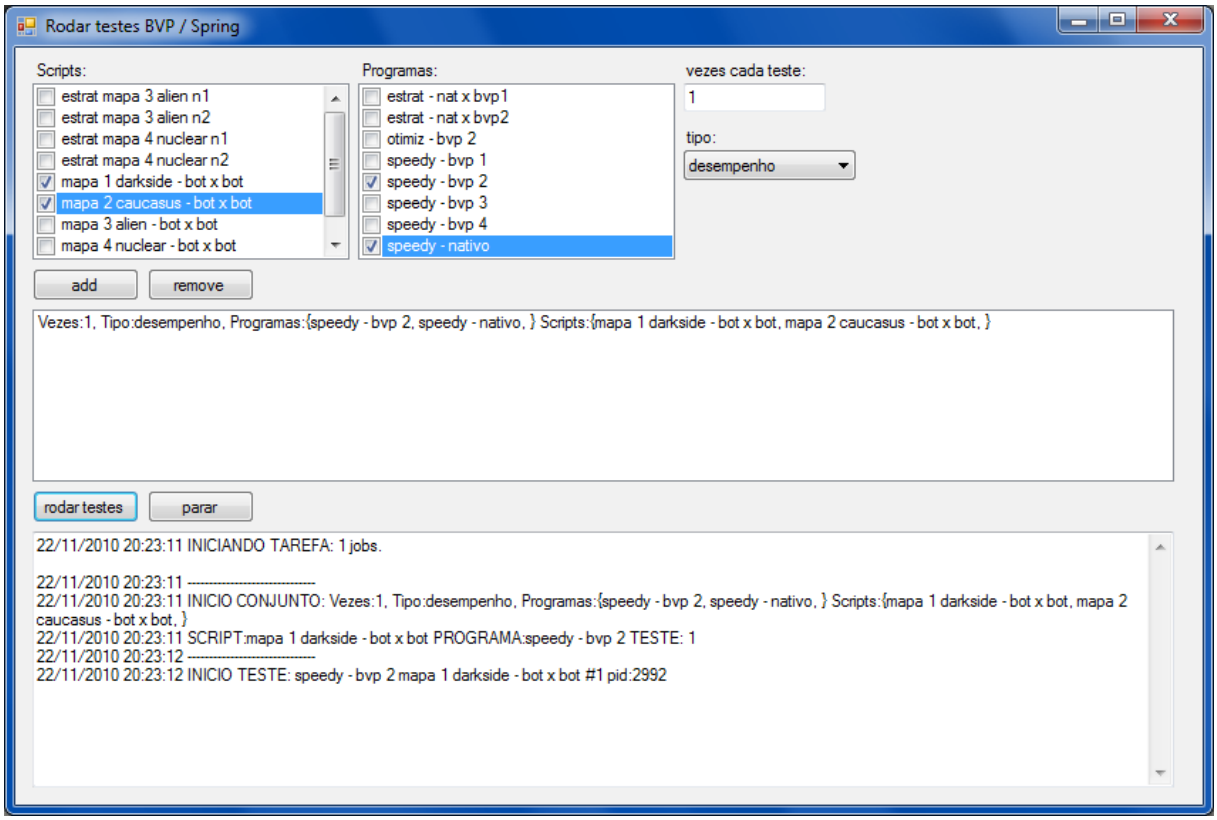


Figura 5.44: Programa gerenciador dos testes.

O programa permite a criação de conjuntos de teste, contendo as configurações de *scripts* de jogo, que informam os mapas e as inteligências artificiais, bem como permite a escolha das diferentes compilações de arquivo executável da Spring contendo configurações diferentes do planejador de caminhos. O programa cuida da execução em sequência dos testes que compõem os conjuntos, incluindo a inicialização da *Spring* com a versão correta de configuração para cada teste, e a leitura e armazenamento do arquivo contendo as estatísticas de jogo ao final de cada execução de um teste pela *Spring Engine*, permitindo ainda que os testes sejam executados em sequência sem interação do usuário, o que possibilita que muitos testes sejam rodados de forma autônoma e consumindo o mínimo de gerenciamento e tempo humano.

Os dois programas auxiliares foram desenvolvidos na linguagem C#, o código fonte foi disponibilizado sob licença GPL, e estão disponíveis na mesma página de Internet juntamente com a implementação do planejador BVP.

5.8.2 Animações de jogo

Alguns vídeos de exemplo foram capturados durante a realização dos testes, e também se encontram disponíveis para acesso via Internet no seguinte endereço:

`http://www.youtube.com/view_play_list?p=9D43A597D1F4987C`

5.8.3 Publicação de artigo

Durante o andamento deste trabalho, os resultados e testes preliminares também geraram material útil que foi aproveitado para a publicação de um artigo relacionado (SIL 2010), do

qual o autor deste trabalho faz parte também como coautor.

O artigo publicado tem como objetivo organizar em uma única publicação as diversas técnicas em estudo, referências já publicadas, e também resultados de estudos experimentais à respeito da pesquisa feita pelo Grupo de Computação Gráfica do Instituo de Informática da Universidade Federal do Rio Grande do Sul na área de planejamento de caminhos usando campos potenciais.

O artigo faz um apanhado das diversas técnicas desenvolvidas pelo grupo para uso no movimento de agentes virtuais, incluindo uma explanação do planejador de caminhos usando campos potenciais que foi objeto de estudo deste trabalho, tratamento de grupos de agentes, interação da técnica com jogos de RTS, melhorias e otimizações usando GPU, resultados experimentais da implementação em um jogo de RTS[22], e conclusões gerais à respeito das técnicas propostas.

Uma cópia completa do artigo em questão pode ser encontrada ao final deste trabalho, no anexo da página 86.

[22] Os resultados comentados dizem respeito à implementação feita neste trabalho.

6 CONCLUSÃO

O desenvolvimento deste trabalho teve como principal objetivo mostrar os resultados da implementação de um planejador de caminhos usando campos potenciais dentro de um jogo de estratégia em tempo real. Para isso, foi realizado o estudo dos conceitos envolvidos, incluindo o planejador de caminhos proposto e a *engine* de jogos, e a realização de testes para verificação do funcionamento do algoritmo.

Seguindo o desenvolvimento da implementação do planejador BVP para a *Spring Engine*, 75 testes foram rodados usando diversas configurações, com o objetivo de se verificar o desempenho em termos de velocidade e escalabilidade do novo planejador BVP. Durante os testes, o planejador de caminhos nativo apresentou uma resposta de jogo normal com uma quantidade de mais de 200 unidades em movimento na simulação, enquanto que o planejador BVP apresentou um limite prático de escalabilidade, resultando em uma queda significativa do desempenho do jogo quando o número de unidades atingiu em torno de 30 unidades em movimento.

Além dos testes de desempenho, testes de qualidade foram feitos com o objetivo de se verificar a qualidade gráfica e visual dos caminhos gerados, em termos de naturalidade de comportamento e suavidade do movimento. Os testes mostraram os caminhos suavizados gerados pelo planejador BVP, comparados com os caminhos artificiais e pouco naturais gerados pelo planejador nativo da *Spring*. Foi mostrada também a tendência do planejador nativo de alinhar em fila as unidades de um grupo em movimento, em contraste à tendência do planejador BVP de posicionar as unidades lado a lado quando em movimento em grupo.

Também foram feitos testes de vantagem estratégica, a fim de verificar se os caminhos gerados apresentam algum tipo de vantagem ou desvantagem estratégica durante o jogo, comparado com o planejador de caminhos nativo. O planejador BVP, na sua configuração original, mostrou uma desvantagem de jogo causada por unidades presas dentro das fábricas. A configuração modificada do planejador BVP, que resolve o problema das unidades presas de uma maneira semelhante ao planejador nativo, apresentou um equilíbrio na relação de vitórias e derrotas em disputas contra o planejador nativo, mostrando por fim que o planejador de caminhos BVP não possui vantagens nem desvantagens significativas em relação ao planejador de caminhos nativo.

Após os testes, foi proposta uma modificação no processo de relaxamento do campo potencial do planejador BVP, com o objetivo de tentar aumentar o limite de escalabilidade do planejador. Testes realizados com a implementação otimizada proposta mostraram um aumento significativo de mais de 100% do número limite de unidades em movimento comparado com o limite obtido com o planejador sem otimização.

Quanto ao código fonte da implementação, foi disponibilizado na Internet sob licença GPL

para uso livre por todos os interessados, juntamente com os programas auxiliares de apoio, e também os vídeos demonstrativos dos testes executados. Resultados parciais e preliminares deste trabalho também foram usados para a confecção de um artigo relacionado ao tema de planejamento de caminhos.

O trabalho serviu para realizar uma comparação direta entre o planejador de caminhos BVP, e um planejador de caminhos usando busca em grafos, baseado em A*. Os resultados mostraram que o planejador BVP gera caminhos mais suaves, menos artificiais, e também que seguem mais diretamente ao objetivo, apresentando menos segmentos oscilatórios do que os caminhos gerados pelo planejador nativo, o que em última instância significa que o planejador de caminhos BVP apresenta uma melhor qualidade visual e naturalidade para a simulação do jogo do que o planejador original da *Spring*, ou de qualquer outro jogo que utilize métodos de planejamento de caminhos que não possuam formas de suavização ou naturalização dos caminhos. Também foi mostrada uma possível vantagem estratégia, que seria o fato das unidades evitarem andar em fila e procurarem andar lado a lado. Entretanto, o teste de vantagem estratégica descartou essa hipótese, porém como os testes foram efetuados com jogadores controlados pela inteligência artificial, ainda existe a possibilidade de um jogador humano explorar com sucesso essa característica à seu favor.

Todavia, a geração de caminhos suaves e naturais fornecidos pelo planejador BVP possui um custo computacional alto em relação aos caminhos gerados pelo planejador nativo, posto que na situação de teste sem otimização, o planejador BVP apresentou uma limitação de escalabilidade. Em vista disso, é imperativo que o planejador BVP utilize algum tipo de otimização, seja ela no processo de relaxamento conforme proposto neste trabalho, seja usando aceleração de desempenho via execução do planejamento na GPU, paralelização do relaxamento usando várias *threads* ou *cores* do processador, ou qualquer outro tipo de método que aumente o limite do número de unidades em movimento.

Para finalizar este trabalho, seguem algumas sugestões de melhorias ou trabalhos futuros, à fim de complementar os estudos efetuados:

- Testes com jogadores reais: os testes com jogadores artificiais são muito subjetivos e não representam com precisão as situação reais de uso do planejador. Testes podem ser conduzidos com vários jogadores reais, e através da aplicação de questionários, as tendências, preferências e impressões podem ser levantadas. A própria questão da vantagem estratégia das unidades andarem lado a lado, ou então a naturalidade dos caminhos gerados pode ser levada em conta no levantamento.
- Testes em computadores diferentes: neste trabalho todos os testes foram efetuados no mesmo computador disponível para uso pelo autor. Testes em outros computadores podem mostrar limites de escalabilidade, resultados visuais, ou de desempenho diferentes.
- Diferentes configurações do planejador BVP: algumas das configurações do planejador BVP foram fixadas para valores escolhidos experimentalmente, e então os testes foram rodados sem uma variação das configurações fixadas, como por exemplo tamanho do mapa local e tamanho da borda *f-zone*, entre outros. A realização de testes levando em conta uma variação desses valores pode apresentar resultados interessantes e relevantes quanto ao uso do método.

REFERÊNCIAS

(PER 2005) PERUCIA, A. S.; BERTHÊM, A. C.; BERTSCHINGER, G. L.; MENEZES, R. R. C. **Desenvolvimento de Jogos Eletrônicos: Teoria e Prática**, São Paulo, Novatec, 2005 p.152-159

(WOO 2002) WOOLDRIDGE, M. J. **An Introduction to MultiAgent Systems**, West Sussex, England, John Wiley, 2002 p. 259

(SIL 2009) SILVEIRA, R.; DAPPER, F.; PRESTES, E.; NEDEL, L. **Natural steering behaviors for virtual pedestrians**. 2009 (http://dx.doi.org/10.1007/s00371-009-0399-0)

(FER 1999) FERBER, J. **Multi-Agent Systems: an introduction to distributed artificial intelligence**, Harlow, England, Addison-Wesley, 1999

(RUS 2004) RUSSEL, S. J.; NORVIG, P. **Inteligência Artificial**, Rio de Janeiro, Campus, 2004

(KUF 1998) KUFFNER, J. J. Goal-directed navigation for animated characters using realtime path planning and control, **Workshop on Modeling and Motion Capture Techniques**, Springer-Verlag 1998 p.171-186

(KAV 1996) KAVRAKI, L.; SVESTKA, P.; LATOMBE, J. C.; OVERMARS, M. H. Probabilistic roadmaps for plath planning in high-dimensional configuration space, **IEEE Trans. on Robotics and Automation**, 1996 p.566-580

(CHO 2003) CHOI, M. G.; LEE, J.; SHIN, S. Y. Planning biped locomotion using motion capture data and probabilistic roadmaps., **ACM Trans. Graph**, 2003 p.182-203

(PET 2002) PETTRE, J.; SIMEON, T.; LAUMOND, J. Planning human walk in virtual environments, **Int. Conf. on Intelligent Robots and System**, 2002 p.3048-3053 vol.3

(TEC 2001) TECCHIA, F.; LOSCOS, C.; CONROY, R.; CHRYSANTHOU, Y. Agent behaviour simulator (abs): A platform for urban behavior development, **Game Technology**, 2001

(DEL 2008) DELISLE, R. K. Beyond A*: IDA* and Fringe Search, **Game Programming Gems 7**, Boston, USA Charles River Media 2008 p.289-294

(TRE 2006) TREVISAN, M. et al Exploratory Navigation Based on Dynamical Boundary Value Problems, **Journal of Intelligent & Robotic Systems**, Netherlands Springer Netherlands 2006 p.101-114 v45

(FIS 2008) FISCHER, L. G. **Otimização de desempenho em planejadores de caminho usando campos potenciais**, Porto Alegre, 2008 63p Universidade Federal do Rio Grande do Sul. Instituto de Informática.

(SIL 2010) SILVEIRA, R.; FISCHER, L.; FERREIRA, J. A. S.; PRESTES, E.; NEDEL; L. Path-Planning for RTS Games Based on Potential Fields, **Motion in Games Third International Conference**, Utrecht, The Netherlands Springer Berlin 2010

(UNI 2010) **UNITY: Game Develpoment Tool** (http://unity3d.com/) última visita em 30/10/2010

(UNR 2010) **Unreal Technology** (http://www.unrealtechnology.com/) última visita em 30/10/2010

(SPR 2010) **The Spring Project** (http://springrts.com/) última visita em 07/11/2010

(IRR 2010) **Irrlicht Engine: A free open source 3d engine** (http://irrlicht.sourceforge.net/) última visita em 30/10/2010

(STR 2010) **STandalone REproducible FLOating-Point library** (http://nicolas.brodu.numerimoire.net/en/programmation/streflop/index.html) última visita em 08/11/2010

ANEXO: ARTIGO PUBLICADO

A seguir pode ser encontrada uma cópia do artigo relacionado publicado (SIL 2010).

Path-Planning for RTS Games Based on Potential Fields

Renato Silveira, Leonardo Fischer, José Antônio Salini Ferreira, Edson Prestes, and Luciana Nedel

Universidade Federal do Rio Grande do Sul, Brazil,
{rsilveira, lgfischer, jasferreira, prestes, nedel}@inf.ufrgs.br,
Home page: http://www.inf.ufrgs.br

Abstract. Many games, in particular RTS games, are populated by synthetic humanoid actors that act as autonomous agents. The navigation of these agents is yet a challenge if the problem involves finding a precise route in a virtual world (path-planning), and moving realistically according to its own personality, intentions and mood (motion planning). In this paper we present several complementary approaches recently developed by our group to produce quality paths, and to guide and interact with the navigation of autonomous agents. Our approach is based on a BVP Path Planner that generates potential fields through a differential equation whose gradient descent represents navigation routes. Resulting paths can deal with moving obstacles, are smooth, and free from local minima. In order to evaluate the algorithms, we implemented our path planner in a RTS game engine.

Keywords: Path-planning, navigation, autonomous agent

1 Introduction

Recent advances in computer graphics algorithms, especially on realistic rendering, allow the use of synthetic actors visually indistinguishable from real actors. These improvements benefit both the movie and game industry which make extensive use of virtual characters that should act as autonomous agents with the ability of playing a role into the environment with life-like and improvisational behavior. Despite a realistic appearance, they should present convincing individual behaviors based on their personality, moods and desires. To behave in such way, agents must act in the virtual world, perceive, react and remember their perceptions about this world, think about the effects of possible actions, and finally, learn from their experience [7].

In this complex and suitable context, navigation plays an important role [12]. To move agents in a synthetic world, a semantic representation of the environment is needed, as well as the definition of the agent initial and goal position. Once these parameters were set, a path-planning algorithm must be used to find a trajectory to be followed.

However, in the real world, if we consider different persons – all of them in the same initial position – looking for achieving the same goal position, each path followed will be unique. Even for the same task, the strategy used for each person to reach his/her goal depends on his/her physical constitution, personality, mood, reasoning, urgency, and so on. From this point of view, a high quality algorithm to move characters across virtual environments should generate expressive, natural, and occasionally unexpected steering behaviors. In contrast, especially in the game industry, the high performance required for real-time graphics applications compels developers to look for most efficient and less expensive methods that produce good and almost natural movements.

In this paper, we present several complementary approaches recently developed by our group [4, 6, 14, 15, 18] to produce high quality paths and to control the navigation of autonomous agents. Our approach is based on the BVP Path Planner [14], that is a method based on the numeric solution of the boundary value problem (BVP) to control the movement of agents, while allowing the individuality of each one.

The topics presented in this article are: (*i*) a robust and elegant algorithm to control the steering behavior of agents in dynamic environments; (*ii*) the production of interesting and complex human-like behaviors while building a navigational route; (*iii*) a strategy to handle the path-planning for group of agents and the group formation-keeping problem, enabling the user to sketch any desirable formation shape; (*iv*) a sketch based global planner to control the agent navigation; (*v*) a strategy to deal with the BVP Path Planner in GPU; (*vi*) a RTS game implementation using our technique.

The remaining of this paper is structured as follows. Section 2 reviews some related works on path-planning techniques that generate quality paths and behaviors. Section 3 explains the concepts of our planner and how we deal with agents and group of agents. Section 4 proposes an alternative to our global path planner, enabling the user interaction. Improvements in the performance are discussed in Section 5. Some results, including a RTS game implementation using our technique, is shown in Section 6. Section 7 presents our conclusions and some proposals for future works.

2 Related Work

The research on path-planning has been extensively explored on the games domain where the programmer should frequently deal with many autonomous characters, ideally presenting convincing behavior. It is very difficult to produce natural behavior by using a strategy focusing on the global control of characters. On the other hand, taking into account the individuality of each character may be a costly task. As a consequence, most of the approaches proposed in computer graphics literature do not take into account the individual behavior of each agent, compromising the planner quality.

Kuffner [8] proposed a technique where the scenario is mapped onto a 2D mesh and the path is computed using a dynamic programming algorithm like

Dijkstra. He argues that his technique is fast enough to be used in dynamic environments. Another example is the work developed by Metoyer and Hodgings [11], that proposes a technique where the user defines the path that should be followed by each agent. During the motion, the path is smoothed and slightly changed to avoid collisions using force fields that act on the agent.

The development of randomized path-finding algorithms – specially the PRM (Probabilistic Roadmaps) [10] – allowed the use of large and more complex configuration spaces to efficiently generate paths. There are several works in this direction [3, 13]. In most of these techniques, the challenge becomes more the generation of realistic movements than finding a valid path. Differently, Burgess and Darken [2] proposed a method based on the *principle of least action* which describes the tendency of elements in nature to seek the minimal effort solution. The method produces human-like movements through realistic paths using properties of fluid dynamics.

Tecchia et al. [16] proposed a platform that aims to accelerate the development of behaviors for agents through local rules that control these behaviors. These rules are governed by four different control levels, each one reflecting a different aspect of the behavior of the agent. Results show that, for a fairly simple behavioral model, the system performance can achieve interactive time. Treuille et al. [17] proposed a crowd simulator driven by dynamic potential fields which integrates global navigation and local collision avoidance. Basically, this technique uses the crowd as a density field and constructs, for each group, a unit cost field which is used to control people displacement. The method produces smooth behavior for a large amount of agents at interactive frame rates. Based on local control, van den Berg [1] proposed a technique that handles the navigation of multiple agents in the presence of dynamic obstacles. He uses a concept, called *velocity obstacles*, to locally control the agents with few oscillation.

As mentioned above, most of the approaches do not take into account the individual behavior of each agent, his internal state or mood. Assuming that realistic paths derive from human personal characteristics and internal state, thus varying from one person to another, we [14] recently proposed a technique that generates individual paths. These paths are smooth and dynamically generated while the agent walks. In the following sections, we will explain the concepts of our technique and the extensions implemented to handle several problems found in RTS games.

3 BVP Path Planner

The BVP Path Planner, originally proposed by Trevisan et al. [18], generates paths using the potential information computed from the numeric solution of

$$\nabla^2 \, p(\mathbf{r}) \; = \; \epsilon \mathbf{v} \cdot \nabla p(\mathbf{r}) \; , \tag{1}$$

with Dirichlet boundary conditions. In Equation 1, $\mathbf{v} \in \Re^2$ and $|\mathbf{v}| = \mathbf{1}$ corresponds to a vector that inserts a perturbation in the potential field; $\epsilon \in \Re$ corresponds to the intensity of the perturbation produced by $\mathbf{v}$; and $p(\mathbf{r})$ is the

potential at position $\mathbf{r} \in \Re^2$. Both $\mathbf{v}$ and ϵ must be defined before computing this equation. The gradient descent on these potentials represents navigational routes from any point of the environment to the goal position. Trevisan et al. [18] showed that this equation does not produce local minima and generates smooth paths.

To solve numerically a BVP, we can consider that the solution space is discretized in a regular grid. Each cell (i, j) is associated to a squared region of the environment and stores a potential value $p(i, j)$. Using Dirichlet boundary conditions, the cells associated to obstacles in the real environment store a potential value of 1 (*high potential*) whereas cells containing the goal store a potential value of 0 (*low potential*).

A high potential value prevents the agent from running into obstacles whereas a low value generates an attraction basin that pulls the agent. The relaxation method usually employed to compute the potentials of free space cells is the Gauss-Seidel (GS) method. The GS method updates the potential of a cell c through:

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \tag{2}$$

where $\mathbf{v} = (v_x, v_y)$, and $p_c = p_{(i,j)}$, $p_b = p_{(i,j-1)}$, $p_t = p_{(i,j+1)}$, $p_r = p_{(i+1,j)}$ and $p_l = p_{(i-1,j)}$. The potential at each cell is updated until the convergence sets in.

After the potential computation, the agent moves following the direction of the gradient descent of this potential at its current position (i, j),

$$(\nabla \mathbf{p})_c = \left(\frac{p_l - p_r}{2}, \frac{p_b - p_t}{2} \right).$$

In order to implement the BVP Path Planner, we used global environment maps (one for each goal) and local maps (one for each agent). The global map is the Global Path Planner which ensures a path free of local minima, while the local map is used to control the steering behavior of each agent, also handling dynamic obstacles.

The entire environment is represented by a set of homogeneous meshes $\{\mathcal{M}_k\}$, where each mesh $\mathcal{M}_k$ has $L_x \times L_y$ cells, denoted by $\{\mathcal{C}^k_{i,j}\}$. Each cell $\mathcal{C}^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $\mathcal{P}^k_{i,j}$. The potential associated to each cell is computed by GS iterations (Equation 2) and then used by the agents to reach the goal. In order to delimit the navigation space, we consider that the environment is surrounded by static obstacles.

3.1 Dealing with Individuals

Each agent a_k has one local map m_k that stores the current local information about the environment obtained by its own sensors. This map is centered in the current agent position and represents a small fraction of the global map. The size of the local map influences the agent behavior. A detailed analysis of the influence of the size of the local map on the behavior of the agent can be found in [14].

The local map is divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*); and the border zone (*b-zone*), as shown in Figure 1. Each cell corresponds to a squared region, similar to the global map. For each agent, a goal, a particular vector $\mathbf{v}_k$ that controls its behavior, and a ϵ_k should be stated. If $\mathbf{v}_k$ or ϵ_k is dynamic, then the function that controls it must also be specified.

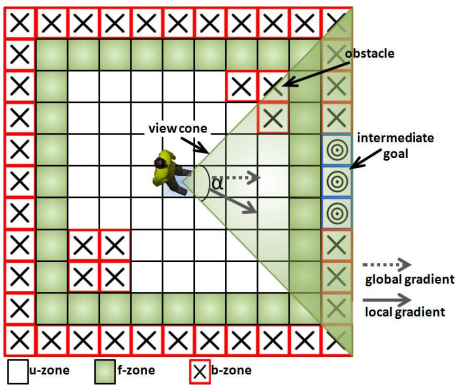


Fig. 1. Agent Local Map. White, green and red cells comprise the *update*, *free* and *border zones*, respectively. Blue, red and light blue cells correspond to the intermediate goal, obstacles and the agent position, respectively.

To navigate into the environment, an agent a_k uses its sensors to perceive the world and to update its local map with information about obstacles and other agents. The agent sensor sets a view cone with aperture α. Figure 1 sketches a particular instance of the agent local map. The *u-zone* cells that are inside the view cone and correspond to obstacles or other agents have their potential value set to 1. Dynamic or static obstacles behind the agent do not interfere in its future motion.

For each agent a_k, we calculate the global descent gradient on the cell of the global map containing its current position. The gradient direction is then used to generate an intermediate goal in the border of the local map, setting the potential values to 0 of a couple of *b-zone* cells, while the other *b-zone* cells are considered as obstacles, with their potential values set to 1. The intermediate goal helps the agent a_k to reach its goal while allowing it to produce a particular motion.

F-zone cells are always considered free of obstacles, even when there are obstacles inside. The absence of this zone may close the connection between the current agent cell and the intermediate goal due to the possible mapping of obstacles in front of the intermediate goal. When this occurs, the agent gets lost because there is no information coming from the intermediate goal to produce a path to reach. *F-zone* cells handle the situation, always allowing the propagation of the information about the goal to the cells associated to the agent position.

After the sensing and mapping steps, the agent k updates the potential value of its map cells using Equation 2 with its pair $\mathbf{v}^k$ and ϵ^k. Hereinafter, it updates its position according to:

$$\Delta\, \mathbf{d} = \upsilon\, (\cos(\varphi^t), \sin(\varphi^t))\, \Psi(|\varphi^{t-1} - \zeta^t|) \tag{3}$$

with

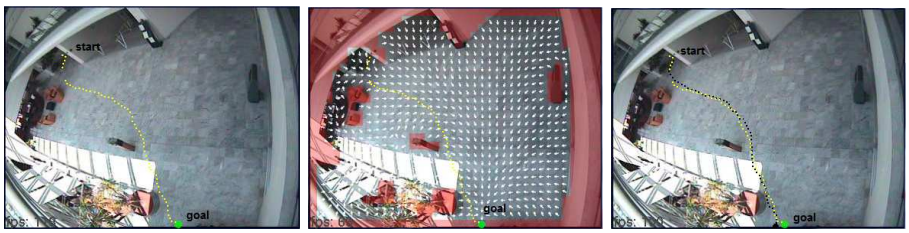
$$\varphi^t = \eta\, \varphi^{t-1} + (1 - \eta)\, \zeta^t \tag{4}$$

where υ defines the maximum agent speed, $\eta \in [0, 1)$, φ is the agent orientation and ζ is the orientation of the gradient descent computed from the potential field stored on its local map in the central position. Function $\Psi : \Re \rightarrow \Re$ is

$$\Psi(x) = \begin{cases} 0 & \text{if } x > \pi/2 \\ \cos(x)\ , \text{otherwise} \end{cases}$$

If $|\varphi^{t-1} - \zeta^t|$ is higher than $\frac{\pi}{2}$, then there is a high hitting probability and this function returns the value 0, making the agent stops. Otherwise, the agent speed will change proportionally to the collision risk.

In order to demonstrate that the proposed technique produces realistic behaviors for humanoid agents, we compared the paths generated by our method with real human paths [14] . Associating specific values to the parameters ϵ and $\mathbf{v}$ in the agent local map, the path produced by the BVP Path Planner almost mimics the human path. Figure 2 shows a path generated by dynamically changing ϵ and $\mathbf{v}$. Up to the half of the path, the parameters $\epsilon = 0.1$ and $\mathbf{v}= (0, -1)$ were used. Half the path forward, the parameters were changed to $\epsilon = 0.7$ and $\mathbf{v}= (1, 0)$. These values were empirically obtained. We can visually compare and observe that the calculated path is very close to the real one.



(a) (b) (c)

Fig. 2. Natural path generated by the BVP Path Planner: (a) the environment with the real person path (yellow); (b) the environment representation discretized by our planner; (c) the agent path (in black) calculated by our planner after adjusting the parameters.

3.2 Dealing with Groups

The organization of agents in groups has two main goals: to allow the control of groups or armies by the user, and to decrease the computational cost through the

management of groups instead of handling individuals. In a previous work [15], an approach to integrate group control in the BVP Path Planner was proposed. This approach also support the group formation-keeping while agents move following a given path. Kallmann [9] recently proposed a path planner to efficiently produce quality paths that could also be used for moving groups of agents. Our technique, illustrated in Figure 3, focus on the group cohesion and behavior while enabling the user to interact with the group.

In this approach, each group is associated to a specific formation and a map, called *group map*. The user should provide – or dynamically sketch – any desired shape for the group formation. This formation is then discretized into a set of points that can be seen as attraction points associated to agents – one point attracting each agent towards him. Analogous to an agent local map, the group map is then projected into the environment and its center is aligned with the center of the group of agents in the environment. The center of the formation shape is also aligned with the center of the group map.

Obstacles and goals are mapped to this map in the same way that we have done for the local maps. However, in the group map there is no view cone. Each agent in a group is mapped to its respective position on the group map as an obstacle. In order to obtain the information about the proximity of an agent in relation to the obstacles, we divide the group map into several regions surrounding the obstacles. Each cell in a region has a scalar value that is used to weight the distance between an agent and an obstacle. When the agent is in a cell associated to any of these regions, it will be influenced not only by the force exerted by the formation, but also by the gradient descent computed at its position in the group map. After that, the vector field is extracted and the agent motion is then influenced by two forces: formation force, and the *group map* vector field. As both forces influence in the path definition, they should be properly established, in order to avoid undesired or non realistic behaviors (for more details on combining these forces, refers to [15].

The motion of the agents which are inside the group map is established using these forces while the motion of the entire group is produced by moving the group map along the global map. For this, we consider the group map center point as a single agent and any path planner algorithm can be used to obtain a path free of collisions.

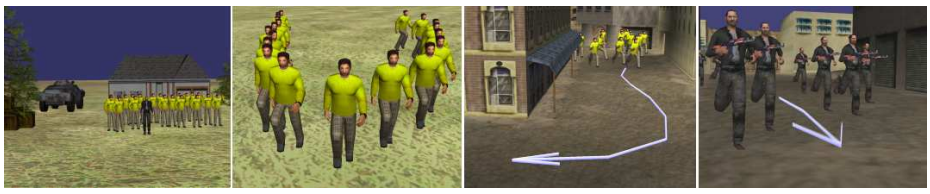


Fig. 3. Group control: agents can keep a formation or move freely; the user can interact and sketch trajectories to be followed by the groups.

4 User Interaction with RTS Games

As seen in Section 3, our technique uses a global map to make a global path-planning, and local or group maps to avoid the collision with dynamic elements of a game (e.g. units, enemies, moving obstacles). This fits very well to most RTS games, where the player selects some units and click on a desired location in order to give a "go there" command.

This kind of game-play commonly requires several mouse clicks to give a specific behavior. RTS players commonly want that a group of units overcome an obstacle by following a specific path that conforms to his/her own strategy. Define a strategy in a high level of abstraction is generally hard to be done with only a few mouse clicks.

Based on the ideas of a previous work from our group [5], we suggested an interaction technique where the units are controlled by a sketch based navigation scheme, as an alternative to the global map . The player clicks with the mouse on the screen and draws the desired path for the currently selected units. The common technique of just clicking on the goal location is also supported. This way of sketching the path to be followed is the same one used in paint-like applications, and can be easily adapted to touch screen displays, like Microsoft Surface®and Apple iPad®, for instance.

4.1 Basic Implementation

The technique is divided in two steps: an *input capture* step, where the user draws the path to be followed; and a *path following* step, where the army units run to their goals. In the first step, the path can be drawn by dragging the mouse with a button pressed, or by dragging his/her finger over a touch screen surface. When the user releases the mouse button or removes his/her finger from the screen surface, a list of 2D points is taken and projected against the battlefield, resulting in a list of 3D points.

In the second step, the points generated on the first step are put in a list and used as intermediary goals for the units. Following this list, each point is used as the position goal for all selected units. When the first unit reaches this goal, it is discarded and the next one in the list is used. This continues until the first unit of the group reaches the last goal.

4.2 Splitting the Army

Imagine a situation where the player has walking soldiers and tanks available to attack, and the enemy headquarters is protected by one of these sides by a swamp. Only walking soldiers can walk through it. Then, the player may choose to attack the enemy by one side with the tanks, and the protected side with the walking soldiers.

An extension of the sketching technique may let the player use this strategy by simply drawing one path to the enemy headquarters, where in a certain division point the path divides into two parts: one goes through the swamp and

the other avoids it. Then, all units (walking soldiers and tanks) follow the path, and in the division point, the army divides into two groups, one that can trespass the swamp and another that goes through mainland.

In order to allow this maneuver, we suggested a structure where the user sketches are stored in a tree. In this tree, each node represents a division point, start point or end point of the motion, and the links between nodes store one section of the path drawn by the player. As in the Section 4.1, each path section is stored as a list of 3D points.

When the player draws the first path, a node a is created for the 3D point where the path starts, and a node b where the path ends. A link storing all the points between a and b is created, with b being child of a. Then, the user draws a second path, starting nearly the point of division chosen by the player along the link. Considering a tree composed by the previously drawn paths, we search for the link l with the path section that contains the closest point to the start point of the newly drawn path. This link l is broken in two parts, l_1 and l_2, and a new end node p is created between both. Finally, we create a new end node c, a link between p and c, and attach the recently drawn path section to the tree.

5 Improvements and Speedup using GPU

Solving the Laplace equation is the most expensive part of the BVP Path Planner algorithm. The iterative convergence process of an initial solution to an acceptable solution demands several iterations. So, if we want to improve the convergence speed, we must focus our attention in the relaxation process. We present an approach to improve the convergence on the local maps based on a GPU implementation.

The technique presented here is highly parallelizable, mainly the update of local maps and the computation of Laplace's equation. Each of these steps has several computations to be done, and it can be accomplished in parallel for each agent. We proposed an approach to implement it using nVidia CUDA®and will present it here assuming that the reader knows the CUDA architecture (a detailed explanation can be found in [6]).

Intuitively, each agent detects its observable obstacles in parallel, and each one has its own objective. So, the update of each local map can be also done in parallel. In our algorithm, each agent must seek for obstacles in its own view cone, setting the corresponding cells in the local map as "blocked". Note that, for a given agent, all other agents are seen as obstacles too. We assume that, in the beginning of this step, all the space occupied by the local map is free of obstacles. Then, for each agent, we launch one thread for each obstacle that the agent can potentially see. In each thread, we check if the obstacle is inside the view cone and inside the local map. If it is, then the thread sets each one of the cells that contains part of the obstacle with a tag "blocked".

This scheme of launching one thread for each obstacle of each agent fits very well in the nVidia CUDA architecture, where the processing must be split into blocks of threads. Assigning one block for each agent, each thread of the block

(a) (b)

Fig. 4. Two autonomous army fighting (a). The unit local map and the path produced by the BVP Path Planner, illustrated by green dots (b).

can look for one obstacle. Also, each thread can update the local map without synchronization, because all the threads will, if needed, update one cell from a "free" state to a "blocked" state.

After the update of the local maps with the obstacles position, we need to set the intermediate goal. For this, we simply need one thread per agent, each one updating the cells with a "goal" tag. This must be done sequentially to the previous step, to avoid race conditions and the need to synchronize all threads of each block of threads. When each local map has up-to-date information about what cells are occupied, free, or goals, we must relax it to get a smooth potential field. To do this, we assigned one local map to one block of threads in CUDA. In each block of threads, each thread is responsible for updating a value of potential to a single cell. Each thread stays in loop, with one synchronization point between the cells at the end of each iteration. Each thread updates the potential value of the cell using the Jacoby relaxation method.

Finally, to avoid unnecessary memory copies between the GPU and the main memory, we store each attribute of all agents in one single structure of contiguous memory. With this, some parameters that do not change so frequently (e.g. the local map sizes, and the current goal) can be sent only once to the GPU. When the new agent position must be computed, we fetch from the GPU only the current gradient descent on the agent positions.

With our GPU-based strategy we achieved a speed up to 56 times the previous CPU implementation. For a detailed evaluation of our results, refers to [6].

6 A RTS Game Implementation using the BVP Path Planner

In order to demonstrate the applicability of the proposed technique, we implemented the BVP Path Planner in the Spring®Engine for RTS games. Spring is an open source and multi-platform game engine available under the GPLv2 license to develop RTS games. We have chosen this engine since it is well known in the RTS community and there are several commercial games made with it and available for use. With our planner, we can populate a RTS game with hundred of agents at interactive frame rates [6]. Figure 4(a) shows a screenshot of the game where two army are fighting using the BVP Path Planner. Figure 4(b) shows one unit and its local map with 33×33 cells. Red dots represent blocked cells, while the yellow dots represent free cells. The path followed by the unit is illustrated by green dots. An executable demo using the BVP Path Planner implemented with the Spring Engine can be found at `http://www.inf.ufrgs.br/~lgfischer/mig2010`, as well as a video including examples that demonstrate all the techniques presented in this paper. All animations in the video were generated and captured in real time.

7 Conclusion

This paper presented several complementary approaches recently developed by us to produce natural steering behaviors for virtual characters and to allow interaction with the agent navigation. The core of these techniques is the BVP Path Planner [14], that generates potential fields through a differential equation whose gradient descent represents navigation routes. Resulting paths are smooth, free from local minima, and very suitable to be used in RTS games.

Our technique uses a global and a local path planner. The global path planner ensures a path free of local minima, while the local planner is used to control the steering behavior of each agent, handling dynamic obstacles. We demonstrated that our technique can produce natural paths with interesting and complex human-like behaviors to achieve a navigational task, when compared with real paths produced by humans. Dealing with groups of agents, we shown a strategy to handle the path-planning problem while keeping the agent formations. This strategy enables the user to sketch any desirable formation shape. The user can also sketch a path to be followed by agents replacing the global path planner.

We also described a parallel version of this algorithm using the GPU to solve the Laplace's Equation. Finally, to demonstrate the applicability of the proposed technique we implemented the BVP Path Planner in a RTS game engine and released an executable demo available on the Internet.

We are now developing a hierarchical version of the BVP Path Planner that spends less than 1% of the time needed to generate the potential field using our original planner in several environments. We are also exploring strategies to use this planner in very large environments. Finally, we are also working on the generation of benchmarks for our algorithms.

References

1. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.: Interactive navigation of multiple agents in crowded environments. In: Proc. of the symposium on Interactive 3D graphics and games. pp. 139–147. ACM (2008)
2. Burgess, R.G., Darken, C.J.: Realistic human path planning using fluid simulation. In: Proc. of Behavior Representation in Modeling and Simulation (BRIMS) (2004)
3. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. 22(2), 182–203 (2003)
4. Dapper, F., Prestes, E., Nedel, L.P.: Generating steering behaviors for virtual humanoids using bvp control. Proc. of CGI 1, 105–114 (2007)
5. Dietrich, C.A., Nedel, L.P., Comba, J.L.D.: A sketch-based interface to real-time strategy games based on a cellular automaton. In: Game programming gems 7. pp. 59–67. Charles River Media, Bostom (2008)
6. Fischer, L.G., Silveira, R., Nedel, L.: Gpu accelerated path-planning for multi-agents in virtual environments. SBGames 0, 101–110 (2009)
7. Funge, J.D.: Artificial Intelligence For Computer Games: An Introduction. A. K. Peters, Ltd., Natick, MA, USA (2004)
8. James J. Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: Int. Workshop on Modeling and Motion Capture Techniques for Virtual Environments. pp. 171–186. Springer-Verlag (1998)
9. Kallmann, M.: Shortest paths with arbitrary clearance from navigation meshes. In: Symposium on Computer Animation (SCA) (2010)
10. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Trans. on Robotics and Automation 12(4), 566–580 (1996)
11. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. Visual Comput. 20(10), 635–649 (2004)
12. Nieuwenhuisen, D., Kamphuis, A., Overmars, M.H.: High quality navigation in computer games. Sci. Comput. Program. 67(1), 91–104 (2007)
13. Pettre, J., Simeon, T., Laumond, J.: Planning human walk in virtual environments. In: Int. Conf. on Intelligent Robots and System. vol. 3, pp. 3048 – 3053 (2002)
14. Silveira, R., Dapper, F., Prestes, E., Nedel, L.: Natural steering behaviors for virtual pedestrians. Visual Comput. (2009)
15. Silveira, R., Prestes, E., Nedel, L.P.: Managing coherent groups. Comput. Animat. Virtual Worlds 19(3-4), 295–305 (2008)
16. Tecchia, F., Loscos, C., Conroy, R., Chrysanthou, Y.: Agent behaviour simulator (abs): A platform for urban behaviour development. In: Proc. Game Technology, 2001 (2001)
17. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. In: ACM SIGGRAPH. pp. 1160–1168. ACM Press, New York, NY, USA (2006)
18. Trevisan, M., Idiart, M.A.P., Prestes, E., Engel, P.M.: Exploratory navigation based on dynamic boundary value problems. J. Intell. Robot. Syst. 45(2), 101–114 (2006)