UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Simulação de heurísticas de pathfinding em cenários de jogos de estratégia em tempo real

Rafael Santos Amorim

Natal-RN

Dezembro de 2015

Rafael Santos Amorim

Simulação de heurísticas de pathfinding em cenários de jogos de estratégia em tempo real

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador

Charles Andryê Galvão Madeira

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Dezembro de 2015

Monografia de Graduação sob o título *Simulação de heurísticas de pathfinding em cenários de jogos de estratégia em tempo real* apresentada por Rafael Santos Amorim e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

_____

Charles Andryê Galvão Madeira
Orientador
IMD
UFRN

_____

Anne Magaly de Paula Canuto
DIMAp
UFRN

_____

Bruno Motta de Carvalho
DIMAp
UFRN

Natal-RN, 10 de Dezembro de 2015.

Agradecimentos

Agradeço a Deus por cada dia me motivar mais e mais para realização dos meus sonhos, me dando sempre força, confiança e esperança. Agradeço à minha mãe Luzinete por sempre se preocupar com minha educação e meu caráter. A minha irmã Rafaela que sempre me deu um bom exemplo e incentivo acadêmico, ajudou bastante para a conclusão desse curso. Ao meu pai Mazinho, que não esteve presente pessoalmente na minha graduação, mas que enquanto esteve neste mundo, me incentivou bastante e sempre acreditou nos seus filhos, e que até hoje está ao meu lado sempre. A todos os familiares, que desde criança fizeram todo o possível para minha dedicação aos estudos. A todos os amigos que fizeram parte dessa caminhada, estudando juntos para provas e trabalhos durante todo o curso. Muito obrigado também aos amigos e namorada que souberam me divertir quando precisei, mas que respeitaram minhas ausências quando precisei me dedicar mais aos trabalhos e provas, além de sempre me darem bons exemplos também.

Agradeço a todos os professores que durante toda caminhada se dedicaram de alguma forma para contribuir com essa formação. Ao professor Charles, que me orientou neste trabalho, sempre cobrando muito e me ensinando bastante. Agradeço ao pessoal da SINFO por todo conhecimento que me passaram desde que cheguei, também contribuindo bastante para a minha formação. Enfim, agradeço a todos que de alguma forma fizeram parte dessa conquista. Obrigado!

“Meus filhos terão computadores, sim, mas antes terão livros. Sem livros, sem leitura, os nossos filhos serão incapazes de escrever - inclusive a sua própria história.”

Bill Gates

Simulação de heurísticas de pathfinding em cenários de jogos de estratégia em tempo real

Autor: Rafael Santos Amorim
Orientador: Charles Andryê Galvão Madeira

RESUMO

Com a evolução dos jogos eletrônicos, o realismo se encontra cada vez mais presente nessas aplicações. É natural, então, que os problemas que a sociedade enfrenta estejam cada vez mais presente nos jogos. Nota-se particularmente que o problema da busca por melhores caminhos, bem conhecido pelo termo *pathfinding* no contexto dos jogos, se faça cada vez mais necessário nesse meio. Vale salientar que a soluções de *pathfinding* nem sempre buscam o caminho mais curto. Muitas vezes o personagem pode escolher um caminho mais seguro, mesmo sendo mais longo, ou outra configuração de caminho que mais lhe agrade de acordo com a estratégia por ele adotada. Este trabalho tem como objetivo analisar o *pathfinding* aplicado ao cenário de jogos de estratégia em tempo real (RTS) através da aplicação de diversas heurísticas e estratégias. Foi concebido um modelo de heurística, baseado no algoritmo A*, que é capaz de calcular o *pathfinding* de acordo com o cenário simulado e as preferências do designer de personagens. Este modelo foi implementado e experimentado no contexto de um cenário baseado no jogo Defense of the Ancients (DotA), utilizando diversos personagens com características diferentes. Os resultados obtidos com os experimentos demonstram a grande variabilidade de possibilidades que cada personagem pode obter nos seus caminhos de acordo com as suas características e com as características do cenário simulado.

Palavras-chave: Pathfinding, Heurísticas, A*.

Simulation of pathfinding heuristics in real-time strategy game scenarios

Autor: Rafael Santos Amorim
Orientador: Charles Andryê Galvão Madeira

ABSTRACT

With the evolution of video games, the realism is more and more present in this kind of application. So, it is natural that problems facing society today are more present in games. In this context, it is particularly noted that searching for better paths is a real problem present in games, well known for the term pathfinding in the gaming context, makes it to become more necessary in this enviroment. It is important to point that the pathfinding solutions do not always find the shortest route. Characters often can choose a safer way, even if it is longer, or another configuration of path that suits according to the strategy adopted. This work aims to analyze the pathfinding problem in the context of real-time strategy games (RTS) by applying several heuristics and strategies. We conceived a heuristic model based on the A* algorithm, which is able to calculate the pathfinding according to the simulated scenario and the preferences of the character designer. This model was implemented and experimented in the context of a Defense of the Ancients (DotA) scenario with several characters composed by different characteristics. The results obtained from the experiments demonstrate a great variety of possibilities that each character can get in its paths according to their characteristics and the characteristics of the simulated scenario.

Keywords: Pathfinding, Heuristics, A*.

Lista de Figuras

Lista de Tabelas

Sumário

1 INTRODUÇÃO

A cada dia que passa, a sociedade se torna mais dependente da tecnologia. A humanidade tem utilizado cada vez mais recursos tecnológicos para resolver problemas do cotidiano. Um desses problemas consiste na busca do melhor caminho para ir de um determinado local a outro. Hoje em dia, na maioria dos smartphones, existe aplicativos capazes de calcular o melhor caminho entre dois pontos em um mapa. Esse problema é conhecido como *Pathfinding* (Rabin, 2002).

O caminho buscado no contexto do *pathfinding* não é necessariamente o mais curto. Dependendo das condições do usuário, pode ser o mais rápido, o mais seguro, etc. Muitas vezes esse *pathfinding* é calculado de maneira tão simples e tão frequente que nem percebemos, como por exemplo, na nossa própria residência, quando nos movemos de um local para outro, e procuramos o melhor caminho. O problema também pode possuir uma complexidade maior, como o caminho que uma pessoa faz diariamente entre sua casa e o seu trabalho. Esse pode ser o mais curto, com menor fluxo de veículos, menos perigoso ou simplesmente mais agradável. Esses critérios são definidos de acordo com as preferências de cada um.

Desde a segunda metade do século XX até os dias atuais, houve um enorme crescimento em várias áreas da computação. Dentre elas se destaca a de desenvolvimento de jogos eletrônicos, que por sua vez não para de crescer (Mendes, 2006). Hoje, os jogos têm alcançado resultados incríveis, com gráficos e realismo cada vez melhores. Por este motivo, percebe-se também que os problemas da vida real estão cada vez mais presente nesses jogos, inclusive o problema do *pathfinding*.

1.1 Contexto

Em muitas competições, milhões de dólares são distribuídos em premiação (Taylor, 2012), e em muitas vezes, elas podem ser decididas com pequenos detalhes, como por exemplo, decisões de melhor caminho possível. Quando se fala do cenário competitivo, há jogadores com qualidades variadas. Assim como no futebol, muitos jogadores possuem uma inteligência tão grande que compensa a deficiência técnica. Em um jogo de tiro em primeira pessoa ou FPS (First Person Shooters), por exemplo, saber atirar é um fator determinante para o sucesso do jogador, assim como saber chutar a bola é no futebol. Porém muitas vezes um jogador que possui grandes reflexos e uma ótima mira chega a perder na estratégia para um jogador com menos habilidades técnicas. Tão importante quanto saber atirar, é saber se posicionar, saber se

locomover, saber tomar importantes decisões. E isso não se aplica apenas a jogos de tiro, mas também a outros jogos como nos de estratégia em tempo-real ou RTS (Real-Time Strategy), nos jogos de interpretação de papéis ou RPG (Role-Playing game), nos simuladores, etc. Sendo assim, além de um bom jogador profissional precisar praticar bastante a ação do jogo, é necessário também estudar, analisar e saber tomar essas decisões, que geralmente aparecem em um curto período de tempo disponível para o jogador pensar. Com a prática, muitas vezes essas decisões acabam sendo tomadas de forma mais automática, pois o jogador acaba obtendo experiência de outras situações parecidas.

Assim como para os seres humanos, o processo de tomada de decisão para o computador também é de extrema importância. No caso do melhor caminho, o computador pode ter a desvantagem de não ter o lado emocional, tal qual o ser humano, que pode ter a psicologia a seu favor ou não, pois conhecendo o seu adversário, pode considerar suas fraquezas. Porém, a inteligência artificial possui uma grande vantagem em relação ao humano pelo fato dela ser capaz de realizar cálculos mais complexos de forma extremamente mais rápida. Percebe-se então a necessidade do estudo do melhor caminho para jogos eletrônicos, tanto para a indústria de desenvolvimento de jogos quanto para a indústria de competições de jogos eletrônicos.

O problema do *pathfinding* nos jogos está mais presente do que imaginamos (Bourg, 2004). Muitas vezes os jogadores possuem objetivos e estratégias diferentes de acordo com a situação. Na Figura 1 é ilustrada uma situação de um jogo FPS em que um personagem do time vermelho precisa plantar uma bomba na área marcada de vermelho, e há um personagem do time azul fazendo patrulha no caminho.

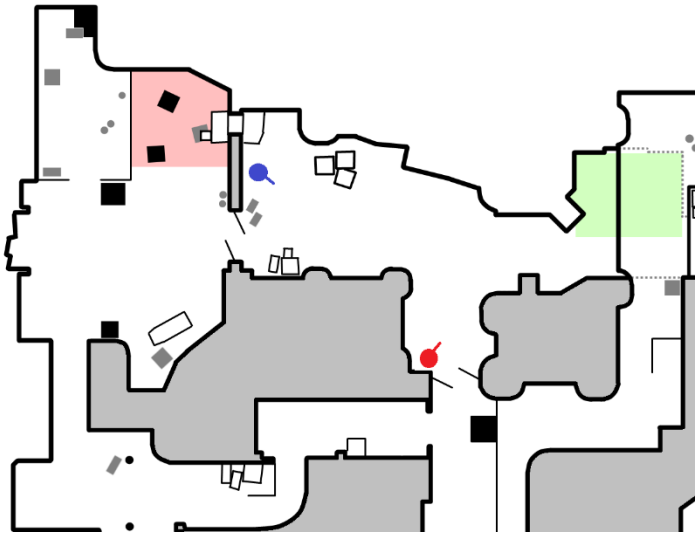

Figura 1: Simulação de uma possível situação de um jogo FPS.

Neste cenário, há diversas caixas que servem como obstáculos, e paredes representadas pelas partes cinzas, nas quais um personagem não pode se locomover. O objetivo do personagem vermelho é plantar uma bomba na área vermelha, e para isso ele tem um certo tempo. Sendo assim, vale mais a pena ele ir pelo caminho mais curto sabendo que terá um inimigo a ser enfrentado? Ou vale mais a pena ele escolher o caminho mais longo, que provavelmente poderá ser mais seguro? Por outro lado, o personagem azul também precisa decidir por qual caminho patrulhar. Sendo assim, podemos perceber que o *pathfinding* não é importante apenas para quem irá se locomover, mas também para quem for tomar uma decisão com base no inimigo.

Outro exemplo que podemos citar são os dos jogos de corrida. Se fosse levado em conta apenas o menor caminho, nas curvas por exemplo, seria escolhido sempre o lado mais fechado da curva. Mas dependendo do veículo e de sua velocidade, esse cálculo poderia dar muito errado. Muitas vezes a eficácia do resultado da curva é melhor se iniciada pelo lado mais aberto, pois mesmo sendo mais longo, permite que o piloto reduza menos a velocidade.

Enfim, o *pathfinding* pode ser encontrado em qualquer tipo de jogo, alguns sendo mais complexos do que outros, e em alguns sendo mais utilizados.

Uma das categorias de jogos eletrônicos que faz bastante sucesso é a dos jogos de estratégia em tempo real (RTS). Dentro desta categoria, estão os jogos que atualmente dominam o mercado mundial, os MOBAs (Multiplayer Online Battle Arena). Os MOBAs mais jogados atualmente, DotA (Defense of the Ancients em www.dota2.com) e LoL (League of Legends em http://br.leagueoflegends.com/), possuem mais de 100 tipos de personagens diferentes.

Nos jogos de RTS, o jogador geralmente se encontra em um cenário, no qual possui o controle de uma ou várias unidades, e essas unidades muitas vezes precisam se deslocar de um local para outro, sendo necessário apenas que o jogador indique o local de destino (Ontanon, 2013). Nesse caso, o sistema do jogo precisa calcular automaticamente o melhor caminho possível. Um dos mais conhecidos jogos de RTS no mundo é a série Age of Empires (http://www.ageofempires.com/), no qual o jogador tem que administrar um império desde o seu início, coletando e armazenando recursos, ao mesmo tempo em que cuida de suas tropas e construções de defesa. Portanto, o jogador pode escolher ações adaptadas a cada um de seus personagens. A Figura 2 mostra uma tela do segundo jogo da franquia, onde podemos perceber que assim como na maioria dos jogos RTS, existe um mini mapa no canto inferior direito da tela para o jogador ter uma visão geral do mundo simulado.

Figura 2: Tela do jogo Age of Empires 2.

No mini mapa, podemos encontrar regiões de cores diferentes que correspondem a áreas nas quais estão posicionadas as unidades dos diversos jogadores. Além disso, há também tipos de terreno diferentes, como grama, árvores, água e pedras. Alguns desses tipos de terreno podem não ser transitáveis por alguns tipos de unidades. Por exemplo, unidades aquáticas somente são capazes de transitar em rios e lagos. Sendo assim, quando um usuário ordena que uma ou mais unidades sigam até um determinado local, elas não necessariamente poderão se movimentar em linha reta. Muito menos se o objetivo for o de escolher o caminho mais seguro, por exemplo.

Encontrar o melhor caminho, é hoje uma das áreas de pesquisa que possui bastante interesse (Algfoor, Sunar, & Kolivand, 2015) pelas empresas de jogos, principalmente no contexto dos jogos RTS. Neste contexto, existem diversos algoritmos que permitem encontrar soluções para resolver o problema do *pathfinding*. Em muitos casos, o *pathfinding* é executado para diversos personagens simultaneamente, e por isso, surge também uma grande preocupação no que se refere ao tempo de processamento.

Na prática, podem existir diversas prioridades. Por exemplo, um personagem pode ser bastante fraco, mas precisar chegar em um local que seu menor caminho passaria por zona(s) de risco. Logo, o seu melhor caminho poderia ser um pouco mais longo, de forma que ele transitasse com mais segurança. Muitas vezes também é necessário considerar o tipo de personagem e o tipo de terreno em questão, pois alguns personagens podem ter mais facilidades

de se locomover em um tipo de terreno do que outro. Essas e outras variações, podem ocorrer nos diferentes estilos de jogos que utilizam *Pathfinding*. Considerando as diversas variações que podem ocorrer no cenário de um jogo, pode existir uma enorme demanda de busca, com suas decisões sendo alteradas em tempo real. Tendo em vista os dados citados, é fácil perceber que o problema do melhor caminho não se resume mais a um simples algoritmo de busca de caminho mais curto. Com tanta variedade de situações e cenários, é necessário muitas vezes que sejam consideradas essas variedades dispostas nos jogos.

1.2 Objetivos

Este trabalho propõe um estudo do problema do *pathfinding* em diversas situações que poderão surgir para os personagens nos jogos de estratégia em tempo real. Soluções para esse problema se fazem cada vez mais necessárias para que o desempenho do comportamento inteligente destes jogos seja satisfatório.

Há uma grande variação de parâmetros a serem avaliados no cálculo do *pathfinding*. Para isso, este trabalho visa construir um modelo capaz de resolver o problema considerando essas variáveis. O modelo proposto neste trabalho, terá como foco um ambiente específico, que são os jogos RTS, mas com a possibilidade de ser adaptado para alguns outros estilos de jogos, como os de FPS.

Para validar esse modelo, um simulador será desenvolvido para permitir ao usuário alterar em tempo real o cenário de jogo, adicionando e removendo personagens, assim como configurando os diversos parâmetros envolvidos.

Esse modelo poderá servir para auxiliar bastante os designers de personagens, uma vez que eles queiram criar um novo personagem, ou até mesmo alterar um já existente, testando e verificando como tal personagem se comportará no ambiente proposto, diante de diversas situações.

1.3 Corpo do Documento

Neste trabalho, pretende-se primeiramente contextualizar o leitor no cenário em que ele está inserido. No capítulo 2, será apresentado com mais detalhes a problemática do *pathfinding* e suas aplicações nos jogos eletrônicos. No capítulo 3, iremos introduzir o conceito de algoritmos de busca como possíveis técnicas para solucionar o problema do *pathfinding*, citando os tipos e alguns exemplos para que o leitor entenda como são calculados. Neste sentido, explicaremos

em detalhes o funcionamento do algoritmo A*, que é a base do modelo proposto neste trabalho. Em seguida vem o capítulo 4, que propõe um modelo heurístico para o cálculo de *pathfinding*, introduzindo os parâmetros que serão tratados no mesmo. No capítulo 5, será apresentado um cenário para realização de testes. Vai ser apresentada a ferramenta desenvolvida para a validação do modelo, um mapa exemplo, os personagens e características envolvidas na metodologia, além de explicar com detalhes alguns testes realizados. No capítulo 6 serão abordadas as considerações finais e os possíveis trabalhos futuros.

2 PATHFINDING

Muitas vezes, na vida real, precisamos tomar uma decisão de *pathfinding* sem nenhum auxilio tecnológico, e dependendo do nosso conhecimento do ambiente, e de experiências anteriores, podemos tomar uma decisão sem fazer muito esforço, e muitas vezes sem mesmo perceber. A maioria das pessoas que possuem um emprego em um local fixo e com horários fixos, sabem qual o caminho mais curto para chegar até suas residências, e sabem também qual caminho possui o maior fluxo de veículos nos diferentes horários do dia. Sem grandes dificuldades, também podem perceber que outros caminhos são melhores quando alteram o modo de transporte. A pé pode ter um caminho bastante diferente de um caminho feito em um veículo pessoal, assim como pode ser bastante diferente do melhor caminho realizado em transportes públicos.

Apesar de toda a capacidade de aprendizado do ser humano, não tem comparação com a velocidade e a precisão de realizar cálculos que um computador possui. Além disso, em muitos casos, o indivíduo pode não conhecer muito bem o cenário do ambiente que precisa ser tratado. É por isso que cada vez mais surgem ferramentas que auxiliam a sociedade no processo de tomada de decisão que visa buscar melhores caminhos.

Uma das ferramentas mais conhecidas e mais utilizadas no mundo é o Google Maps, que além de fornecer informações do cenário, calcula também os caminhos mais curtos para diferentes meios de locomoção.

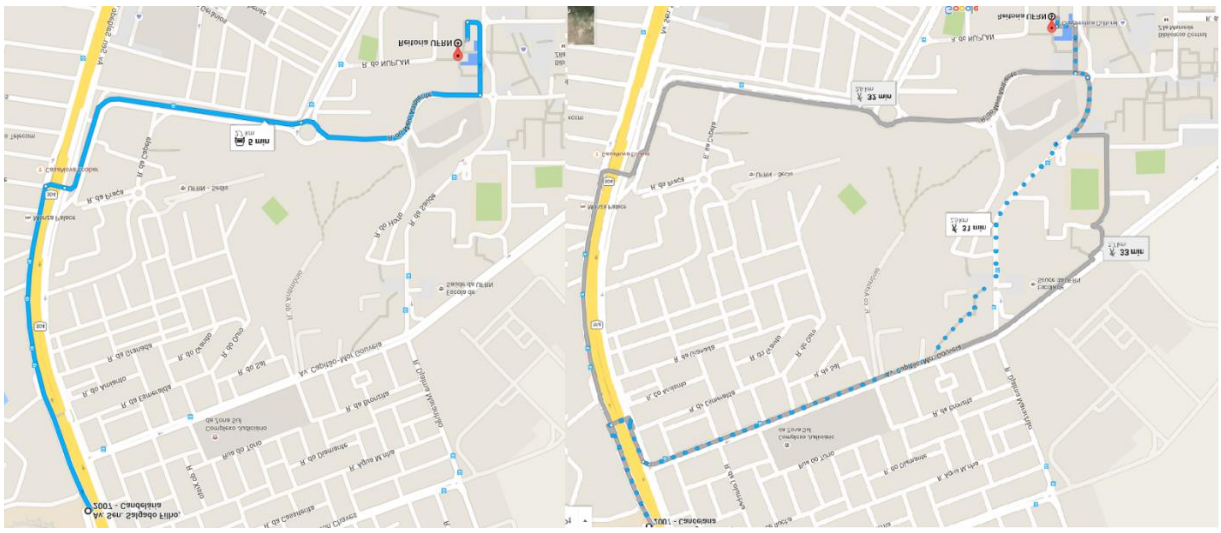

Figura 3: A esquerda uma imagem do Google Maps indicando o melhor caminho de carro. A direita o melhor caminho a pé.

Na figura 3, foi realizada uma busca do melhor caminho na cidade de Natal-RN, partindo da Arena das Dunas até a Reitoria da Universidade Federal do Rio Grande do Norte (UFRN). No lado esquerdo da figura,, foi buscado o caminho utilizando um carro pessoal, enquanto que no lado direito da figura, foi considerado apenas que o indivíduo está a pé. É notável a grande diferença entre as soluções. Esses cálculos foram realizados em uma velocidade extremamente rápida, e de forma bem precisa. Por outro lado, vale salientar que a precisão do resultado depende diretamente da precisão da representação do cenário. Por exemplo, se for criado um atalho, ou uma nova rua e não estiver atualizada no mapa, o ser humano tem a vantagem de saber disso.

Ainda no cenário de uma cidade, mas já entrando na realidade dos jogos eletrônicos, um dos grandes sucessos da empresa Sega é a franquia do jogo Crazy Taxi, no qual o jogador precisa controlar um taxi, buscando e deixando passageiros nos seus destinos, mas tudo sendo realizado em um determinado tempo, que é bem limitado. O sistema do jogo calcula o menor caminho dentro das leis de trânsito para o jogador seguir e o informa em tempo real. Porém, o jogo permite que o jogador tome decisões de seguir um atalho, subir uma calçada e até mesmo atravessar praças públicas. É importante lembrar também que o jogo continua calculando esse caminho a todo momento, ou seja, quando o jogador decide seguir outro caminho, o sistema calcula novamente considerando a nova posição e as novas condições do jogo. Na figura 4, podemos observar uma tela do jogo Crazy Taxi, em que o sistema indica com um seta, qual a direção sugerida ao jogador.

Figura 4:Tela do jogo Crazy Taxi

Na maioria dos jogos, existem diversos personagens que muitas vezes possuem grandes variações de características. Nos diversos estilos de jogos que temos hoje, esses personagens

necessitam se movimentar, e esse movimento vai de acordo com o estilo do jogo. Em um jogo 2D de plataforma, por exemplo, o personagem pode se movimentar geralmente através das plataformas que, as vezes, podem até se deslocar. Em alguns jogos o jogador pode se movimentar indicando a direção onde quer ir ou simplesmente já indicando o próprio destino desejado.

Além da movimentação do personagem controlado pelo jogador, há também a mesma situação para as unidades controladas pela máquina. Seja de um inimigo que quer alcançar um objetivo, ou um personagem que pretende atender um pedido de ajuda de outro personagem, por exemplo.

Em jogos de movimentação mais simples, uma simples máquina de estados pode resolver o problema da movimentação de um personagem controlado pela máquina ou NPCs (Non-Player Characters). Em alguns casos, podem até serem feitas rotas fixas, como no caso de um personagem que está patrulhando uma área. Ou até mesmo uma movimentação aleatória. Nesses casos citados, a solução é bem mais simples, assim como o custo computacional é bem menor. Desse modo existem soluções de sucesso como é o caso do jogo do Mario, maior ícone da Nintendo, que possui diversos personagens automatizados que apenas fazem rotas fixas. Na maioria das vezes, os personagens automatizados deste tipo de jogo seguem rotas fixas de patrulha, implementadas com simples máquinas de estado. Algumas vezes também é possível perceber que o personagem adversário sempre persegue o personagem protagonista que é controlado pelo jogador. Nesse caso, é possível perceber o *pathfinding*, mas obviamente que com uma complexidade bem mais simples dos que nos jogos RTS, pois as possibilidades de movimentação são bem limitadas.

Estas soluções simples muitas vezes não são suficientes para algumas categorias de jogos. No caso dos jogos RTS, é bastante comum existir personagens com características diversas, inclusive na sua movimentação. No mesmo jogo é possível encontrar unidades que só podem se movimentar em determinados tipos de terrenos, outras que podem voar sobre determinados obstáculos, outras com capacidades de movimentação sobre a água, etc. Em muitos casos, também há itens e habilidades dos personagens que os permitem se locomover de forma diferente, possibilitando alcançar locais que suas movimentações normais não permitem.

2.1 Categorias de Pathfinding

O problema do *pathfinding* pode ser dividido em 2 categorias: estático e dinâmico.

O estático é utilizado em situações nas quais o ambiente não sofre alteração, ou seja, o personagem possui as mesmas informações do ambiente durante todo o processo, não havendo alterações que possam influenciar no cálculo. Um exemplo disto é o já citado caso da aplicação de mapas desenvolvida pela Google, no qual o computador tem informações do cenário, e realiza os cálculos necessários sem que ao mesmo tempo ocorram alterações no cenário.

Contudo, na prática, nem sempre isso acontece. Muitas vezes no contexto real dos jogos eletrônicos, diversas alterações de cenário podem acontecer a todo momento. Ao escolher um caminho, o *pathfinding* deve estar apto às mudanças imediatas que podem ocorrer no cenário. Novos adversários podem surgir, novos caminhos e novos bloqueios podem aparecer no caminho. Além disso, outras variáveis podem sofrer alterações. Por exemplo, se em um determinado momento o jogador desejar buscar o caminho mais curto, mas em seguida sofrer um ataque adversário que o deixe bastante debilitado, ele poderá tomar a decisão de mudar para um caminho mais seguro.

2.2 Representação do Ambiente

Para o computador resolver o problema do *pathfinding* em um jogo, o ambiente simulado precisa ser representado de alguma forma que permita o seu tratamento. Um exemplo clássico de representação de possibilidades de movimentação é o de um grafo. O grafo (Tenenbaum, 1995) é uma estrutura de dados que possui vértices (nós), que podem ser conectados entre si através de arestas. Além disso, essas arestas podem ter pesos, que podem ser levados em conta em várias situações. No contexto do *pathfinding*, cada vértice corresponde a um local do terreno, as arestas sendo possíveis caminhos entre estes locais, cada uma com um custo associado. Segue na Figura 5 um exemplo de grafo com pesos nas arestas.

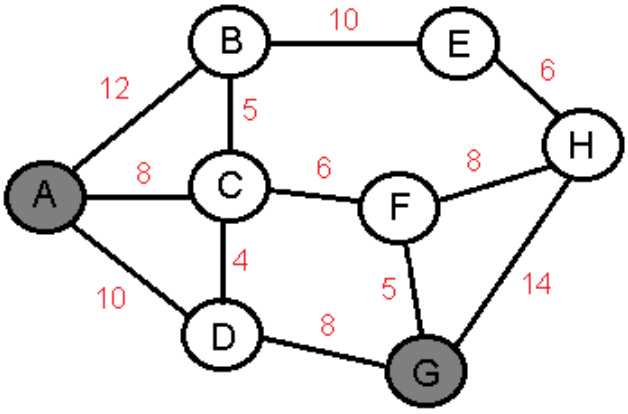

Figura 5: Exemplo de Grafo com peso nas arestas

Os locais representados pelos vértices podem ser organizados de várias formas de acordo com o jogo. Em alguns casos, o terreno pode ser discretizado, formando representações em pequenos retângulos, em triângulos, losangos, dentre outras formas. Quanto menor for essa forma, mais detalhada será a representação do mundo, e consequentemente o resultado pode ser mais realista. Por outro lado, uma grande quantidade de vértices e arestas pode exigir um custo computacional maior. Vale salientar que o terreno não precisa ser necessariamente dividido em partes iguais e simétricas. Na Figura 6, o terreno é dividido em partes iguais, onde cada parte dessa representa um vértice.

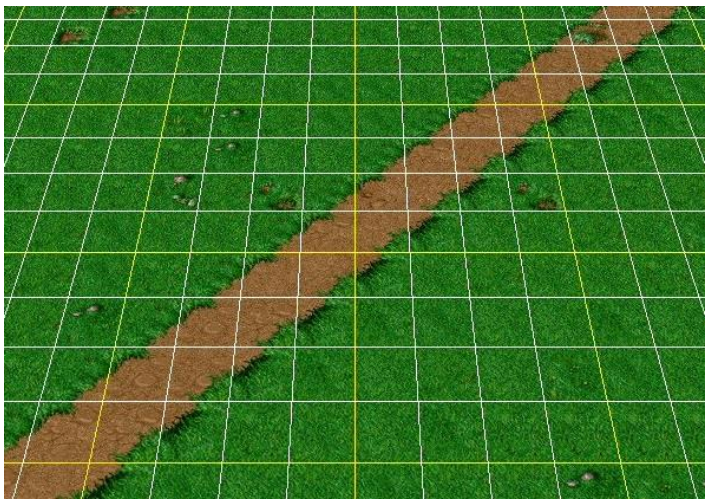

Figura 6: Terreno dividido em pequenas partes

No caso da Figura 6, essa representação é conhecida como grid (Bourg, 2004), que se baseia na utilização de polígonos regulares de mesmo tamanho. Os polígonos mais conhecidos para esta representação são triângulos, quadrados e hexágonos (Red Blob Games, s.d.).

Esse tipo de representação em formato de grid pode ser muito bem adaptada para jogos RTS, porém, pode não ser tão eficaz, por exemplo, em um mapeamento entre cidades para uma aplicação que deseja calcular o *pathfinding* entre cidades. Imagine um cenário com as cidades A e B, interligadas por uma longa estrada. Fazer uma representação em grid neste caso, iria gerar uma enorme quantidade de nós desnecessariamente, aumentando a complexidade de resolução. Esse caso poderia ser facilmente representado por um grafo, no qual cada cidade seria um vértice, e suas estradas seriam as arestas.

Encontrar a solução do *pathfinding* a partir de um grafo consiste em um problema de otimização, e para isto, algoritmos de buscas permitem resolver esses problemas.

3 ALGORITMOS DE BUSCA

Algoritmos de busca (Cormen, Leiserson, & Rivest, 2009) são soluções para problemas de otimização. Para a representação do problema, pode ser utilizado um grafo como já mencionado no capítulo anterior, podendo até mesmo ser uma árvore. A árvore é um grafo que possui um nó raiz, que pode possuir um ou mais filhos, que por sua vez, podem possuir mais filhos, até que cheguem as folhas. As folhas são os nós que não possuem filhos.

Naturalmente, um algoritmo de busca percorre o grafo em busca da solução para o problema, de forma que a cada passo, um determinado nó seja expandido e verificado se este é a solução. Caso não seja, é escolhido um novo nó a ser expandido de acordo com o algoritmo. Alguns desses algoritmos, já tem seu caminho de busca construído por algum critério, como por exemplo, as buscas em largura e em profundidade nas quais um determinado passo não depende do passo anterior. Por outro lado, alguns dependem totalmente do andamento da busca, como exemplo os algoritmos gulosos que sempre escolhem a melhor solução imediata. Assim como os algoritmos que utilizam métodos que estimam qual o provável melhor caminho, fazendo com que a possibilidade de encontrar o melhor caminho aumente.

3.1 Busca Cega

Os algoritmos de busca cega (Cormen, Leiserson, & Rivest, 2009) utilizam apenas as informações fornecidas pela definição do problema. Entre os diferentes algoritmos de busca cega, podem ser adotados diferentes critérios, que para cada situação podem encontrar ou não uma solução mais rapidamente do que outra. Entre os algoritmos de busca cega, podemos citar a busca em profundidade e a busca em largura.

Na busca em profundidade, o algoritmo expande sempre o nó não expandido mais profundo da árvore que ainda não foi expandido, independente dos valores contidos neles (a menos que tenha encontrado a solução). A Figura 7 mostra como o algoritmo da busca em profundidade expande seus nós. Perceba que a cada passo, é buscado o nó mais profundo.

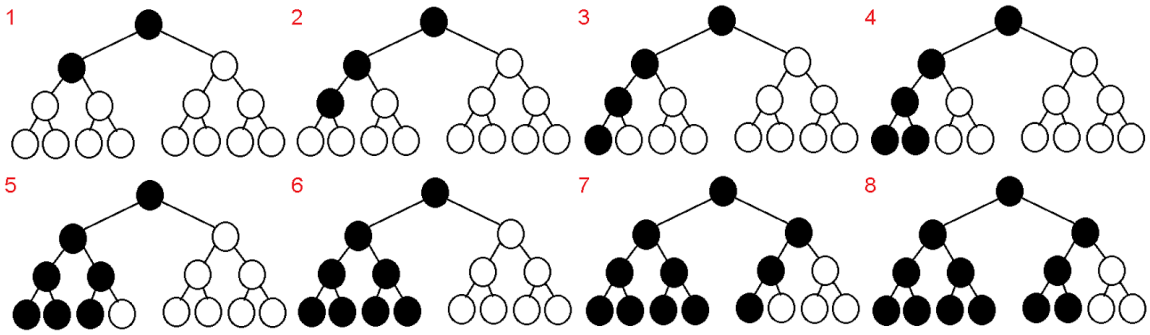

Figura 7: Passo a passo do algoritmo de busca em profundidade.

Inicialmente o algoritmo começa a procurar pela raiz da árvore. No primeiro passo, ele verifica se a raiz possui filho(s), onde no caso apresentado dois são encontrados.. O primeiro filho é expandido e já verificado se ele também possui filho, e assim segue, até que encontre uma folha (passo 3). Quando uma folha é encontrada, o algoritmo volta para o pai dela e se ele tiver outro filho, a busca segue da mesma forma neste filho. Perceba nos passos 6 e 7 que a busca só chega em outro filho da raiz quando todos os sucessores de um filho da raiz são totalmente expandidos.

Assim como a busca em profundidade, outro exemplo de busca cega é a busca em largura. A busca em largura também segue um critério fixo que independe dos valores em cada vértice. A Figura 8 mostra a sequência seguida na busca em largura.

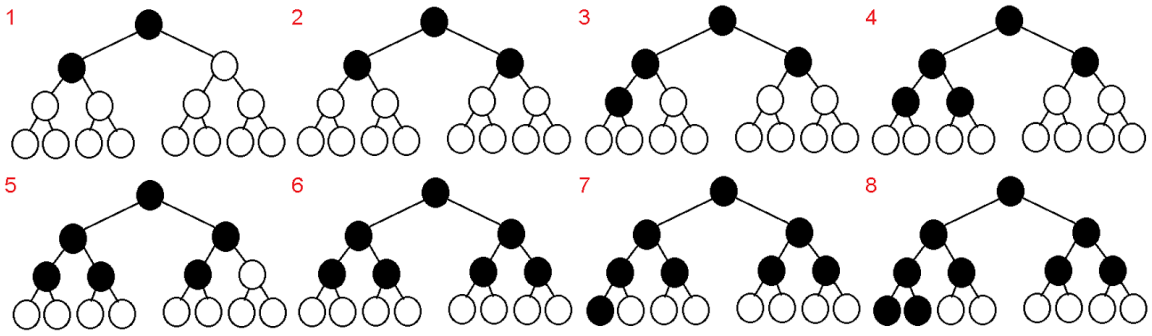

Figura 8: Passo a passo do algoritmo de busca em largura.

No caso da busca em largura, são verificados todos os nós de um mesmo nível, para poder em seguida descer a outro. Perceba entre os passos 2 e 3 que todos os filhos da raiz são expandidos, para em seguida, expandir os netos. O mesmo pode ser observado entre os passos 6 e 7.

Além desses tipos, também há outros tipos de buscas cegas. Esses algoritmos de busca cega, ou exaustiva, costumam ter mais facilidade na sua implementação, no seu entendimento

e além disso, para cada nó verificado, costuma ter um custo computacional menor. Por outro lado essa busca pode ser considerada ineficiente na maioria dos casos. Fazendo uma analogia com o mundo real, imagine que quando acontece um crime em uma cidade, seria mais fácil a polícia começar procurando pistas para investigar os principais suspeitos ou ir logo investigando cada cidadão daquela cidade? Obviamente que há uma possibilidade do culpado ser um dos primeiros investigados da cidade, porém, há uma chance muito maior de encontrá-lo se procurar pistas para direcionar a busca pelo culpado. Da mesma forma acontece nos grafos. Há casos em que buscar de forma exaustiva pode ser melhor, mas na maioria dos casos não é o que acontece. Com isso, temos uma outra categoria de busca conhecida como busca heurística.

3.2 Busca Heurística

A busca heurística (Cormen, Leiserson, & Rivest, 2009) utiliza o conhecimento específico do problema para expandir o seu próximo nó. No caso do crime citado anteriormente, poderia ser considerado quem foi visto no local do crime no dia, para obter mais pistas ou até mesmo encontrar o culpado. Isso gastaria um pouco mais de esforço para decidir qual seria a próxima pessoa a ser investigada, porém, essa pessoa teria mais chances de ser o culpado ou de direcionar melhor a busca.

No algoritmo de uma busca heurística, existe uma função que é calculada a cada vez que será escolhido um novo nó a ser expandido. Esse cálculo torna a escolha mais custosa do que a escolha do próximo nó na busca cega. Contudo, geralmente é mais eficiente, pois na maioria dos casos, essa escolha do próximo nó é feita em uma quantidade bem menor.

A função que ajuda a escolher o próximo nó a ser expandido, pode ser chamada de função heurística. A função heurística tem a finalidade de estimar qual a escolha menos custosa a se fazer entre o estado atual e o estado final mais próximo.

Entrando agora em outro cenário, usando o grafo da Figura 5, é desejado encontrar o caminho de menor custo entre os vértices A e G, considerando que o custo de um vértice para outro sendo o valor que está em vermelho, próximo a cada aresta. Se a busca fosse cega, não importaria os custos informados em cada aresta. Mas como se trata de uma busca heurística, a escolha de cada nó depende diretamente desses valores. Um dos algoritmos da categoria mais fáceis de implementar e de baixo custo computacional, é conhecido como algoritmo guloso.

3.2.1 Algoritmos Gulosos

O algoritmo guloso (Cormen, Leiserson, & Rivest, 2009) busca apenas a melhor solução imediata, tendo como consequência o fato de que nem sempre é encontrada a melhor solução, assim como mostra no grafo da Figura 5. Usando um algoritmo guloso, partindo do vértice A, o primeiro passo seria expandir para o vértice C, pois entre os vizinhos de A, é o que possui o menor custo. Seguindo na mesma lógica, os próximos seriam os vértices D e G respectivamente, obtendo um custo total de 20. Nesse caso, é fácil perceber que o caminho menos custoso seria partindo de A para D e depois para G, que teria um custo total de 18.

Apesar do algoritmo guloso buscar sempre a melhor solução imediata, é comum ver essa ideia em outros algoritmos, que a utilizam em alguns casos. Um dos mais conhecidos algoritmos de busca para essa situação é o algoritmo de Dijkstra.

3.2.2 Dijkstra

O algoritmo de Dijkstra (Cormen, Leiserson, & Rivest, 2009) coloca inicialmente um peso infinito em cada vértice. A cada iteração, são verificados todos os vizinhos do nó que está sendo investigado. No caso da Figura 5, começa pelo vértice A, verificando seus vizinhos B, C e D. Se o peso desse nó, adicionado ao custo da aresta que liga ele a seu vizinho, for menor do que o atual peso do vizinho, esse vizinho terá seu peso atualizado. Para a iteração seguinte pode ser então utilizado o vizinho de menor peso atual, e isso é uma forma de estimar qual poderia ser o caminho de menor custo. Quando o objetivo é encontrado, é verificado então o caminho percorrido para saber qual o de menor custo.

Vários algoritmos surgiram utilizando uma ideia parecida com a de Dijkstra também, assim como podem utilizar a ideia de um algoritmo guloso. O algoritmo A* pode ser considerado uma combinação dos dois. Usa uma ideia bem parecida com o Dijkstra, mas que possui uma função heurística a mais que ajuda na escolha do próximo nó a ser verificado.

3.2.3 Algoritmo A*

A forma tradicional do A* (Cormen, Leiserson, & Rivest, 2009) consiste em calcular a distância através da soma de duas funções, nas quais chamaremos de função G e função H. Para realizarmos a busca, é necessário primeiramente transformar o terreno em um grafo de pesos positivos. São utilizadas duas listas de nós do grafo, nas quais chamaremos de lista aberta e lista fechada. A lista aberta guarda alguns nós que esperam ser verificados, enquanto a lista fechada

armazena os nós que já foram verificados. Inicialmente a lista fechada começa vazia, enquanto a lista aberta começa apenas com o ponto inicial da busca. Primeiramente são adicionados à lista aberta, todos os vizinhos deste ponto inicial em que o tipo de terreno permite a passagem do personagem, sem esquecer de indicar que o ponto inicial é o seu pai. Em seguida, deve-se remover o ponto inicial (o nó que estava sendo verificado) da lista aberta e inserida na lista fechada. Esses passos devem se repetir até que seja encontrado o nó que represente o objetivo final.

Mas em cada passo, qual nó deve ser escolhido para ser verificado? Para isto são usadas as funções citadas anteriormente (G e H). A função G é simplesmente a distância real percorrida para alcançar um determinado nó a partir de um outro. Essa função pode variar de acordo com a forma geométrica escolhida para representar cada parte do terreno. A função H, por sua vez, é um custo estimado do movimento entre o nó atual e o objetivo final. Vale salientar que este custo é apenas uma estimativa, pois não há como saber com precisão ainda, visto que podem haver diversos tipos de terrenos e obstáculos pelo caminho. Esta função H pode ser calculada com diversas técnicas diferentes. A soma das funções G e H resulta na função F.

Após cada nó verificado, este nó é retirado da lista aberta e adicionado na lista fechada. O nó seguinte a ser verificado será o nó com o menor custo F entre os que estão na lista aberta. É importante notar que quando um nó verificar seu vizinho e este já estiver na lista aberta, deve-se comparar a função F que ele já tinha com a que ele terá pelo novo caminho. Se o novo for menor, este deve então ser considerado o novo pai do nó e o novo custo da função F. A busca chega ao fim quando o objetivo final é encontrado. Neste caso, deve-se verificar os pais do objetivo final até chegar no ponto de origem, para formar o caminho. Ou então, a busca pode terminar quando a lista aberta estiver vazia, isto quer dizer que o objetivo final não foi encontrado.

No grafo da Figura 5, ocorre de forma bem parecida com o Dijkstra, porém, na escolha de qual vizinho seria expandido primeiro, seria escolhido o vértice mais próximo do vértice que se deseja chegar. Essa distância se refere às posições em que os vértices estão.

O algoritmo A* é bastante utilizado no contexto dos jogos eletrônicos (Cui & Shi, 2011), tratando principalmente a problemática do *pathfinding*. Além disso, existem (Cormen, Leiserson, & Rivest, 2009) vários desdobramentos do mesmo (A* hierárquico, A* dinâmico, etc.).

No contexto do algoritmo A*, qualquer tipo de heurística pode ser colocada em prática. Portanto, no próximo capítulo será proposto um modelo que engloba as principais variáveis que

podem influenciar a forma de movimentação das unidades nos jogos de RTS.

4 MODELO

Nos capítulos anteriores foram citados diversos estilos de jogos que apresentam de alguma forma o problema do *pathfinding*. Neste capítulo será proposto um modelo que encontre soluções para o problema no contexto dos jogos de RTS. Este modelo é capaz de encontrar soluções de *pathfinding* de acordo com as configurações do cenário e dos personagens, permitindo que os designers de personagens possam testar facilmente diversas formas de como personagens poderão agir no contexto do processo de movimentação.

O modelo utiliza como base a função heurística H do algoritmo A* para calcular o *pathfinding*. Este modelo visa ser genérico, funcionando para qualquer tipo de representação do cenário, e neste capítulo serão apresentadas algumas características que são importantes para ele.

4.1 Função Heurística

Neste modelo, cada parâmetro pode ser levado em conta ou não, de acordo com as preferências do usuário. Os parâmetros que serão considerados, têm seus custos calculados separadamente, e depois são unidos, formando uma expressão matemática que resulta na função heurística.

Cabe ao usuário (designer de personagens) decidir quais os parâmetros são mais importantes para ele, e quais pesos atribuir para cada situação e cada personagem. Assim, para experimentar o modelo, ele usa o seu próprio cenário de jogo, definindo a representação do mundo da forma que ele achar melhor, com um tipo de cálculo de distância associado (Manhattan, euclidiana, etc.).

Esses parâmetros trabalham com custos que são pré-definidos em cada cenário. Cada tipo de terreno tem um determinado custo para cada personagem. Esse custo será adicionado ao valor da função heurística caso o personagem se movimente através dele. Além do tipo de terreno, outro parâmetro a ser avaliado pode ser o relevo. Por exemplo, um personagem pode obter um custo maior ao se locomover por uma subida. Além desses parâmetros, a presença de inimigos em campo pode ser um fator para alteração do resultado, dependendo da estratégia adotada. Todos esses parâmetros serão explicados mais detalhadamente a seguir. A aplicação dos mesmos também será discutida mais na frente. A função heurística pode ser calculada da seguinte forma:

$$H = H1 + M * (H2 + H3 + H4)$$

Neste caso, a função H depende de alguns outros parâmetros. H1 se refere à distância estimada do nó de origem até o alvo desejado. Essa distância pode ser calculada de diversas formas, como será mostrado na seção 4.1.2. O M indica o custo de movimentação de um nó ao seu vizinho, este valor depende bastante do tipo de representação de mapa adotado. Em conjunto com isso, estão os parâmetros H2, H3 e H4, que representam respectivamente o tipo de terreno, a alteração de relevo e o custo de perigo naquele local. Esses são valores que podem interferir diretamente no cálculo da função heurística do modelo desenvolvido. De acordo com as preferências do usuário, alguns desses valores podem nem ser considerados. Por exemplo, se o usuário desejar calcular apenas o caminho mais curto, ignorando adversários, a variável H4 será removida do cálculo. Além desses valores, algumas outras características são levadas em consideração para auxiliar o cálculo desses valores. Nas próximas seções, esses valores serão melhor detalhados.

4.1.1 Personagens

Como já foi explicado neste trabalho, cada tipo de personagem pode possuir características próprias, e essas características podem influenciar diretamente no cálculo do *pathfinding*. Este modelo propõe que um personagem seja facilmente testado pelo designer de personagens, indicando apenas os seus atributos. Esses atributos podem ser características de locomoção, combate, entre outros. Além disso, seus atributos também podem ser alterados com bastante facilidade, para que o usuário consiga realizar seus testes com mais liberdade e perceber o seu comportamento com diferentes valores de parâmetros. Por exemplo, o usuário pode criar personagens voadores, aquáticos, etc.

4.1.2 Distância

Existem diversos métodos matemáticos que calculam a distância entre dois pontos. Esses métodos podem dar uma boa estimativa da distância entre o ponto no qual o algoritmo está verificando e o ponto que se deseja chegar. Lembrando que podem haver diversos obstáculos no caminho, dificultando a precisão dessa estimativa. O método escolhido para calcular é influenciado diretamente pelo tipo de representação de terreno escolhido.

4.1.3 Tipo de Terreno

Cada personagem possui uma série de valores que indicam o custo que o personagem terá ao passar por cada tipo de terreno. Isso quer dizer que para cada tipo de terreno, se o usuário desejar, pode ser considerado custos diferentes para o cálculo da função heurística. Dependendo do cenário proposto, podem haver diversos tipos de terrenos (grama, água, areia, etc.), e de acordo com cada jogo, o designer pode usar os tipos de terreno que desejar.

4.1.4 Relevo

Além do tipo de terreno, o designer de personagens também pode considerar o relevo no cálculo da função heurística. Cada personagem também possui um valor que indica o peso de subida e descida dele, e esse peso será calculado de acordo com a altura no qual ele estará subindo ou descendo.

4.1.5 Estratégia Adotada

O modelo proposto também permite buscar um caminho com uma certa segurança, de acordo com a estratégia adotada. Para utilizar de tal parâmetro, é de extrema importância o bom funcionamento de um algoritmo de visibilidade, pois é ele quem dirá se há inimigos detectados ou não. Cada personagem também possui um valor que indica o seu poder, ou seja, o quanto ele pode assustar. No modelo, são buscados todos os inimigos vistos pelo personagem e adicionados os seus poderes em uma variável. O valor do poder de um personagem é fixo, porém, a forma como ele é adicionado na variável de perigo pode ser diferente dependendo da estratégia adotada. Operações matemáticas podem ser realizadas para diminuir ou aumentar a influência desse valor no custo do caminho. No código a seguir, dois exemplos de como podem ser tratados esses pesos:

```
perigo += Personagem.getPoder(inimigosBloco[i].tipo);
perigo += (int)Mathf.Round (Personagem.getPoder(inimigosBloco[i].tipo)/2);
```

Este cálculo é feito para cada inimigo detectado, e todos eles são somados. Após calcular o perigo em determinado local, ele também é adicionado na expressão matemática que é calculada na função heurística, assim como os outros parâmetros.

4.1.6 Visibilidade

Para simular o campo de visão que o personagem tem, é necessário utilizar um algoritmo de visibilidade. Os algoritmos de visibilidade podem verificar os locais em que o personagem possui visão, e ele pode ocorrer de diversas formas de acordo com a categoria do jogo e a forma na qual o mundo é representado.

Esse algoritmo de visibilidade pode funcionar de diversas formas. Dependendo do jogo, o personagem pode ter uma visão de 360º graus, ou seja, ele consegue ver ao seu redor em todas as direções. Em outros casos, pode ser uma visão em formato de cone. Além disso, os algoritmos podem também considerar obstáculos ou não.

5 METODOLOGIA EXPERIMENTAL

Depois da criação do modelo, é necessário que ele seja validado, e para isto, será apresentado neste capítulo um cenário de um jogo RTS, no qual serão detalhadas as características de terrenos e personagens criados. Em seguida, alguns testes devem ser realizados, utilizando cada parâmetro para fazer diversas possíveis combinações, que podem ser interessantes para um designer de personagens. O objetivo é permitir via uma interface gráfica, todas as possíveis combinações de parâmetros no cálculo da heurística de *pathfinding* no cenário simulado.

Logo, um simulador será desenvolvido com o motor Unity 5 para permitir ao designer de personagens executar todas as formas de experimentos com a função heurística, criando possibilidades de caminhos dos mais variados para permitir facilitar a criação de personagens.

O cenário criado é baseado em um dos grandes sucessos da atualidade, o DotA 2. Essa escolha foi feita por alguns motivos. Primeiro que é um jogo de RTS no qual a necessidade de calcular o *pathfinding* ocorre a partida inteira, além de ser o foco deste trabalho. Além disso, é um dos maiores responsáveis por abrir as portas para o surgimento de outros grande MOBAs no mundo todo, como como LoL e HoN. O DotA 2 também tem sido palco de torneios com premiações milionárias, que podem ser decididos nas escolhas de caminho que o jogador escolheu.

O DotA surgiu apenas como um mapa amador do grande sucesso Warcraft, da Blizzard. Vendo todo o sucesso dos MOBAs, a Valve então resolveu desenvolver o DotA 2. O jogo está sendo um dos grandes responsáveis pela revolução das competições de jogos eletrônicos no mundo, fornecendo sempre as maiores premiações da história, em seus campeonatos. Em seu campeonato mundial de 2015, a competição premiou no total mais de 18 milhões de dólares. Bateu o recorde que já era do mesmo jogo, que no ano anterior teria premiado no total mais de 10 milhões. Além de movimentar a economia dos times, o torneio também tem movimentado bastante o cenário mundial das competições de jogos eletrônicos, que têm lotado estádios pelo mundo inteiro, como mostra na Figura 9.

Figura 9:Campeonato de Dota 2.

No Brasil, apesar de ainda ser pouco valorizado, o cenário tem crescido consideravelmente. Em 2015, foram realizadas algumas partidas de um campeonato nacional de LoL em um dos mais modernos estádios de futebol de mundo. O Allianz Parque, do Palmeiras reuniu mais de 12 mil fãs para o evento.

Considerando todo esse crescimento dos MOBAs e toda a necessidade do *Pathfinding* neles, foi criado neste trabalho um ambiente que simule um MOBA, com o mapa e personagens baseados no DotA 2.

Uma partida comum de Dota 2 é disputada por dois times, que possuem 5 jogadores cada. Durante toda a partida, os jogadores participam de várias batalhas por todo o mapa para ganhar recursos e ficarem mais fortes. O objetivo é destruir uma determinada construção na base adversária que, para chegar lá, o time precisa antes destruir as torres que estão pelo caminho. Muitas vezes acontecem batalhas distantes de onde está um certo personagem, e ele deseja chegar o mais rápido possível. Também pode acontecer de um personagem está extremamente fraco e precisa voltar para a base em segurança.

5.1 O Cenário do Jogo

Como já mencionado, o cenário simulado foi inspirado no jogo DotA 2, que possui um mapa bastante interessante para os testes. Lembrando que se trata de uma inspiração, e não uma cópia fiel. A Figura 10 representa o mapa utilizado atualmente no jogo DotA 2.

Figura 10: Mapa atual do jogo Dota 2.

5.2 Representação do Mundo para o Cálculo da Heurística

Para a representação deste mapa, foi criada uma matriz de duas dimensões, de tamanho 50x50. Apesar de ser uma matriz de duas dimensões, os blocos possuem um atributo que indica o relevo, ou seja, indica a altura em que cada bloco se encontra. Esses blocos têm a forma quadrada. Utilizando sprites disponibilizados gratuitamente na internet, a matriz gera o mapa mostrado na Figura 11. Para representação deste cenário, iremos considerar que cada lado deste quadrado possui 10 unidades de medida, ou seja, a distância do centro de um quadrado até o centro do seu vizinho vertical ou horizontal é 10. Utilizando o método de Pitágoras, podemos calcular que a distância entre o centro de um quadrado e o centro do seu vizinho na diagonal é equivale aproximadamente a 14, que para simplificar, iremos considerar o valor exato de 14. Apesar da escolha ser de uma matriz de tamanho 50x50 para representar o mapa, ele poderia ser melhor detalhado se fosse representado por uma matriz maior, nesse caso, os resultados obtidos seriam mais precisos. Porém, quanto maior o tamanho da matriz que representa o mapa,

maior é também a quantidade de vértices e arestas, sendo assim, também seria maior o custo computacional para a execução do algoritmo.

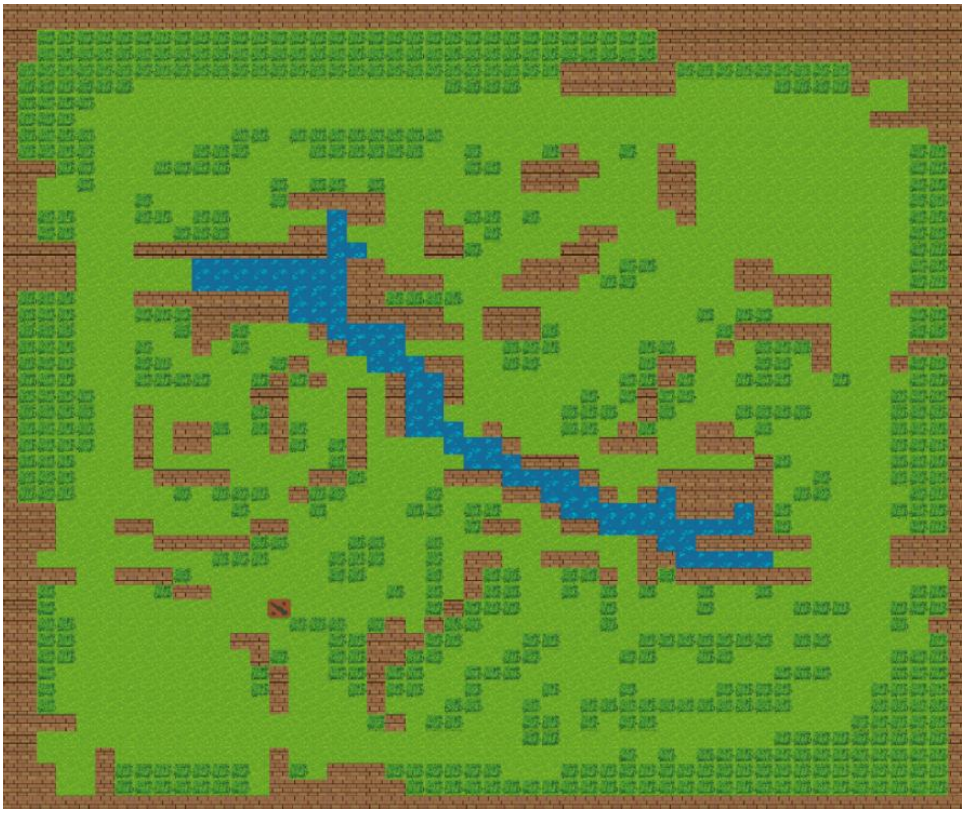

Figura 11: Mapa criado baseado no mapa de Dota 2.

5.2.1 Tipos de Personagens

Além de simular o terreno, também foram criados 7 personagens inspirados em personagens do jogo, que inclusive são representados pelos seus ícones e nomes reais. Cada personagem possui características próprias que serão definidas a seguir:

- **Rubick**: Personagem humano e leve que consegue se locomover normalmente, possui a habilidade de levitar objetos, e pode utilizar isto para saltar sobre paredes. Ele possui habilidades de luta medianas.
- **Sven**: Personagem com corpo similar ao ser humano, porém mais pesado e carrega uma enorme espada, que pode atrapalhar um pouco na sua movimentação. É um personagem com bastante força de ataque.
- **Crystal**: Personagem humano e bastante leve, apesar de ser um personagem terrestre, utiliza muitas habilidades aquáticas. Possui um ataque bastante fraco.

- **Tidehunter**: Personagem gigante que apesar de ser terrestre, é bastante adaptado com a água. Seu ataque também não é muito forte.
- **Drow**: Personagem humano e bastante ágil. Sua velocidade pode ser determinante para a sua movimentações em terrenos complicados e até mesmo para seu poder de ataque.
- **Sladar**: Personagem aquático de porte médio, consegue se locomover pela grama com uma certa dificuldade, mas tem bastante facilidade na água. Possui um ataque mediano.
- **Viper**: Um dragão voador, um personagem que por voar, tem facilidade em qualquer tipo de terreno do cenário proposto, além de ter um bom campo de visão, possui um enorme poder de ataque.

5.2.2 Distância

Para o cenário proposto, o método utilizado para calcular a distância entre dois pontos é a distância de Manhattan, que já foi explicado neste documento. Essa distância pode ser facilmente calculada no cenário proposto, como mostra no método a seguir:

```
int heuristicaManhattan(Bloco b, Bloco t){
        return (Mathf.Abs(b.x - t.x) + Mathf.Abs(b.y - t.y)) * 10;
}
```

Perceba que o método utiliza simplesmente as coordenadas de cada ponto, e aplica na fórmula da distância de Manhattan, que é a diferença da posição vertical somada da diferença da posição horizontal.

5.2.3 Tipo de Terreno do Cenário

Para o cenário proposto, foram definidos 4 tipos de terrenos: grama, água, árvore e parede. E para cada personagem, foram elaborados valores que simulam os custos de movimentação para cada tipo de terreno. Esses valores não são realistas, mas adaptados considerando suas características descritas no capítulo anterior. Esses valores podem ser verificados na Tabela 1.

NOME	GRAMA	ÁGUA	ÁRVORE	PAREDE	SUBIDA	VISÃO	PODER
RUBICK	1	4	3	6	2	5	3

SVEN	1	4	-	-	3	4	6
CRYSTAL	1	3	3	-	2	4	1
TIDEHUNTER	1	2	-	-	3	6	2
DROW	1	4	2	6	1	6	5
SLADAR	3	1	-	-	4	3	4
VIPER	1	1	1	2	2	7	7

Tabela 1: Tabela que indica os atributos de cada personagem do cenário simulado.

Na Tabela 1, são exibidos os valores equivalentes para cada atributo. Nas colunas "GRAMA", "ÁGUA", "ÁRVORE" e "PAREDE" são os custos que cada personagem tem em cada tipo de terreno utilizado, sendo o "-" usado para indicar que é um custo muito alto e não é permitido se locomover por esse tipo de terreno. Note que outros parâmetros estão também descritos na tabela, e eles ainda serão melhor detalhados neste capítulo.

5.2.4 Tipo de Relevo do Cenário

Além do tipo de terreno, cada parte é descrita com o relevo que pode ser levado em conta no cálculo ou não. Cada posição do mapa possui um valor que indica a altura. Além disso, cada personagem tem um valor que indica o peso da subida, que será multiplicado pela diferença entre as alturas. Por exemplo, se o personagem está na altura 2, e deseja subir para um local de altura 3, considerando que ele tem um peso 2 de subida, teremos a seguinte expressão matemática: (3-2) * 2 = 2. Ou seja, 2 será o custo de subida que será adicionado no cálculo da função heurística. Esse valor do peso que cada personagem tem de subida também está descrito na Tabela 1, na coluna "SUBIDA".

5.2.5 Algoritmo de Visibilidade no Cenário

No jogo DotA 2, o jogador tem visão de cima para baixo. Sendo assim, foi desenvolvido um algoritmo de visibilidade para tal característica. No cenário proposto, cada personagem possui um atributo que indica qual o alcance de sua visão, ou seja, a quantidade de blocos que ele consegue ver em linha reta sem obstáculos ou alteração de relevo no caminho. Como estão sendo usados blocos quadrados, para calcular essa distância na diagonal, é considerada a distância de Manhattan, já citada anteriormente. Ou seja, se não houver obstáculos, o personagem terá visão de todos os blocos que estiverem com a distância de Manhattan menor ou igual ao seu alcance de visão. Isto pode ser melhor representado na Figura 12 na qual o

círculo azul indica a posição do personagem e os campos vermelhos indicam a região que o personagem pode ver. Nesse caso o personagem possui o atributo de visão igual a 3.

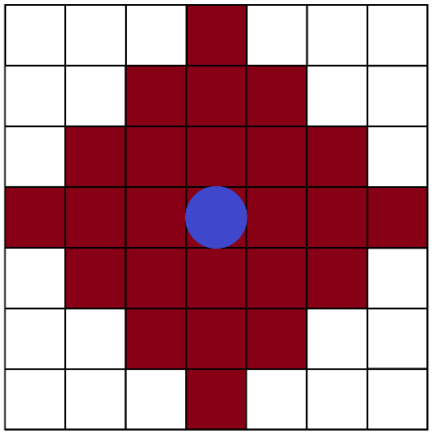

Figura 12: Campo de visão sem obstáculos.

Em conjunto com esse cálculo, também é verificado em todos os lados se há obstáculos, usando a equação da reta. Em todos os lados é lançada a equação da reta para verificar se entre determinado bloco e o personagem há algum obstáculo, para que possa tirar sua visão deste bloco. Segue um exemplo do campo de visão com obstáculos. Agora, com blocos cinzas, que representam paredes.

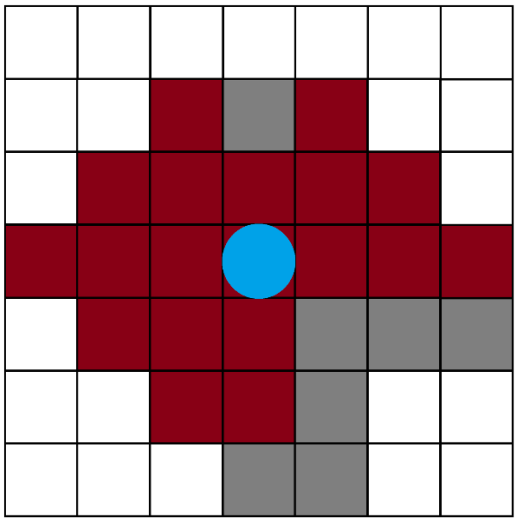

Figura 13: Campo de visão com obstáculos.

5.2.6 Estratégias Abordadas no Cenário

Para o cenário proposto, foram criadas 3 estratégias diferentes: Neutra, Defensiva e Muito defensiva. A estratégia neutra simplesmente calcula o caminho mais curto. A defensiva utiliza o algoritmo de visibilidade já descrito e verifica se há inimigos a vista. Caso positivo, é adicionado um certo peso de acordo com o poder do inimigo encontrado. A estratégia muito defensiva funciona da mesma forma, porém, este peso é maior do que a defensiva.

5.3 Simulador Desenvolvido

Para a realização dos testes, foi desenvolvido um simulador utilizando o Unity 5, com scripts implementados na linguagem C#. Para melhor visualização desses testes, foram utilizados sprites obtidos gratuitamente na internet.

A ferramenta possui algumas opções que permite o usuário customizar seus testes. A seguir, uma imagem da visão geral do simulador desenvolvido.

Figura 14: Interface da ferramenta.

A interface permite que o usuário escolha entre os 7 personagens criados neste cenário. Além disso, note que há algumas opções do lado esquerdo da tela. A primeira opção é a "Diagonal", que possui a finalidade de permitir ou não que o personagem se locomova pela diagonal. Caso contrário, ele só fará movimentos verticais e horizontais. Logo abaixo, temos a

opção de “Considerar Relevo”, que indica se o relevo será considerado ou não no cálculo da função heurística. A opção “Exibir Relevo” é uma funcionalidade apenas visual, quando marcada exibe o relevo do cenário. Em seguida existem 3 botões com as 3 estratégias já mencionadas. O botão “Exibir Visão” também tem funcionalidade apenas para visualização, e quando marcada, é exibida a área que o personagem consegue ver. Por fim, visão com relevo, que quando marcada, calcula a visão do personagem considerando o relevo do cenário.

No simulador, o usuário pode posicionar o personagem na posição em que desejar e será calculado o melhor caminho entre ele e o alvo. É possível posicionar até 5 inimigos pelo terreno, sendo esses inimigos também escolhidos entre os personagens disponíveis, como mostra a figura 15.

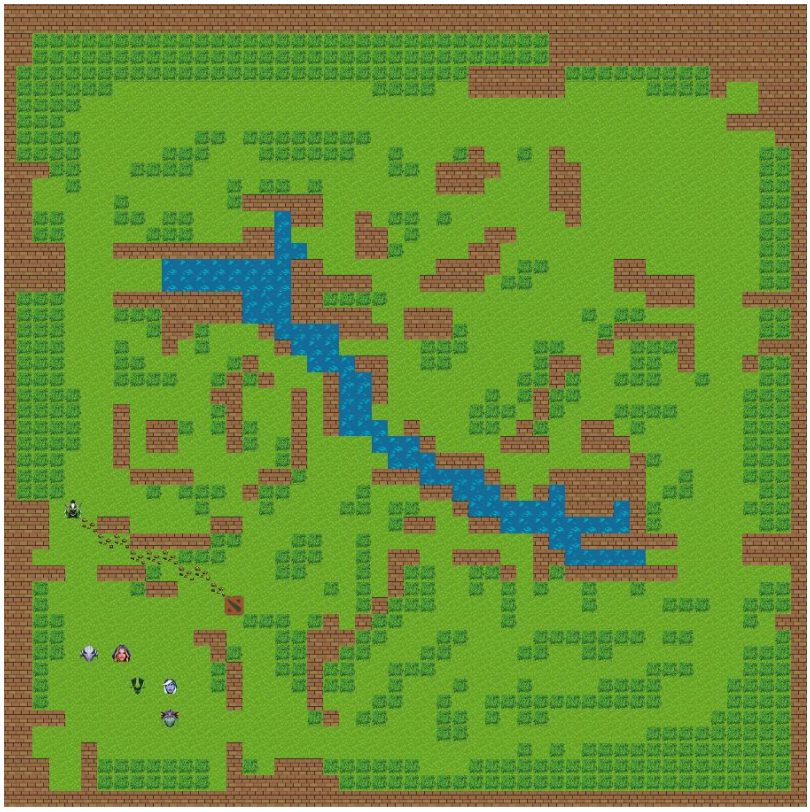

Figura 15: Ferramenta exibindo alguns inimigos.

Se o usuário escolher considerar o relevo para o cálculo do melhor caminho, a ação de um personagem se locomover de um local mais baixo para um mais alto terá um custo associado. A Figura 16 dá uma noção geral do relevo no cenário simulado. Nas duas regiões do mesmo tipo de terreno com cores diferentes, significa que são de alturas diferentes. Por exemplo, o verde da grama á mais escuro quando é mais baixo.

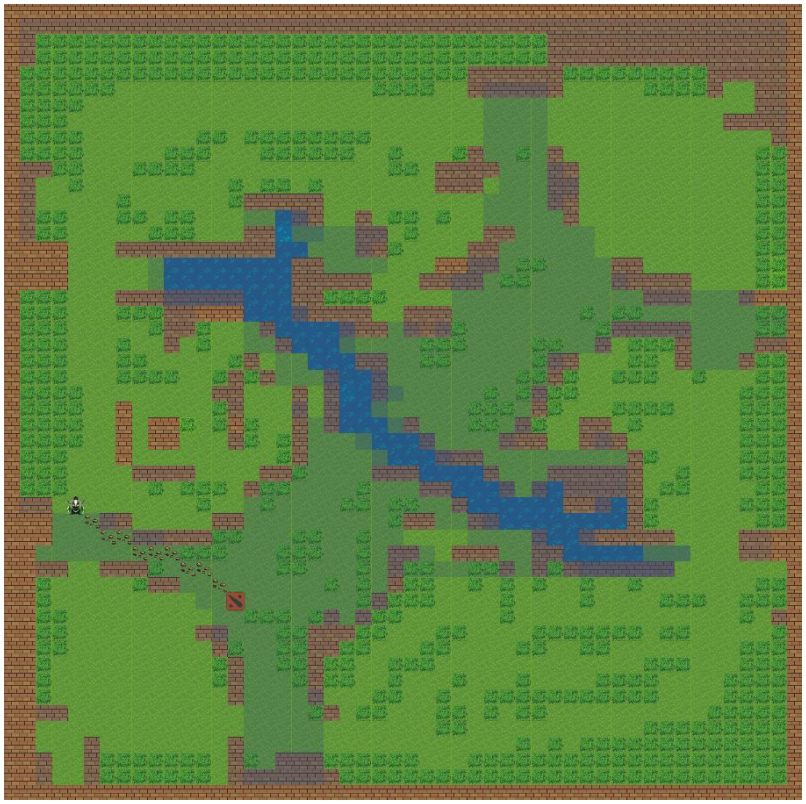

Figura 16: Ferramenta exibindo o relevo do cenário simulado.

Como já explicado, é executado também um algoritmo de cálculo de visibilidade, que tenta simular o campo de visão do personagem de forma simples. O usuário então também tem a opção de visualizar esta área, como mostra nas imagens da Figura 17.

Figura 17: Na imagem da esquerda o campo de visão sem nenhum obstáculo. No meio com alguns obstáculos. Na imagem da direita além de obstáculos, ainda considera o relevo.

Para verificar o comportamento dos personagens em diferentes estratégias, basta o usuário inserir inimigos pelo cenário e escolher uma estratégia. Nas figuras abaixo, está um exemplo de neutro e muito defensivo, no qual é fácil perceber a mudança da escolha.

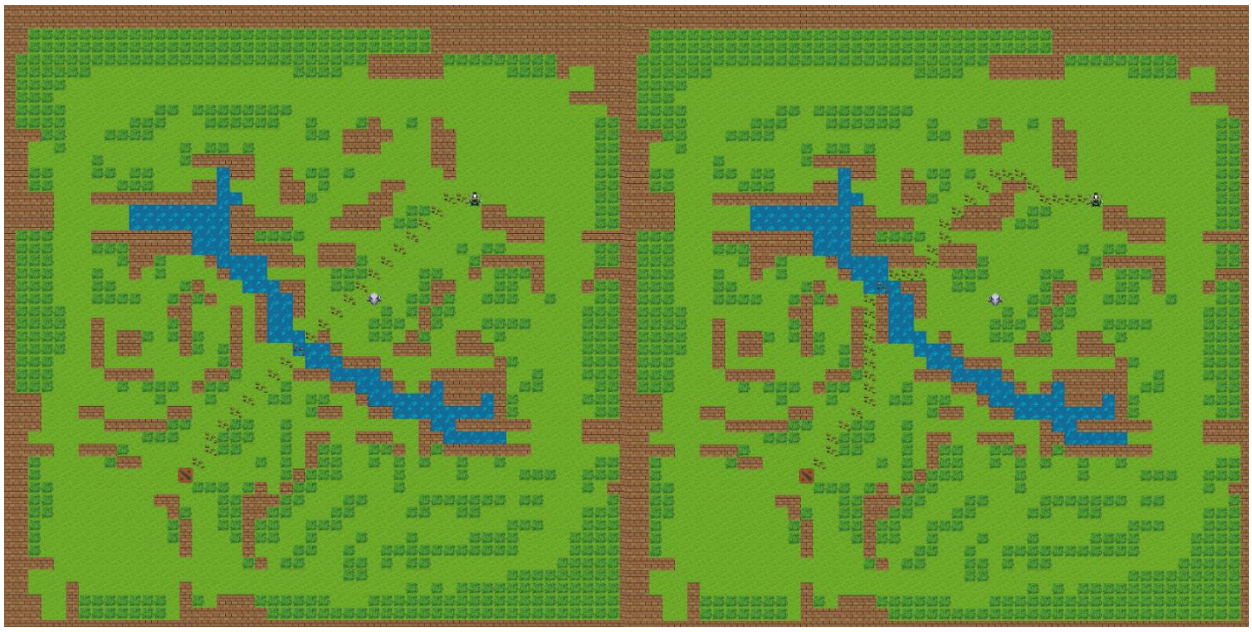

Figura 18: Duas situações iguais com duas estratégias diferentes.

Para auxiliar nos testes, o simulador também permite que até 5 caminhos sejam gravados para serem melhor comparados, como mostra o exemplo da Figura 19.

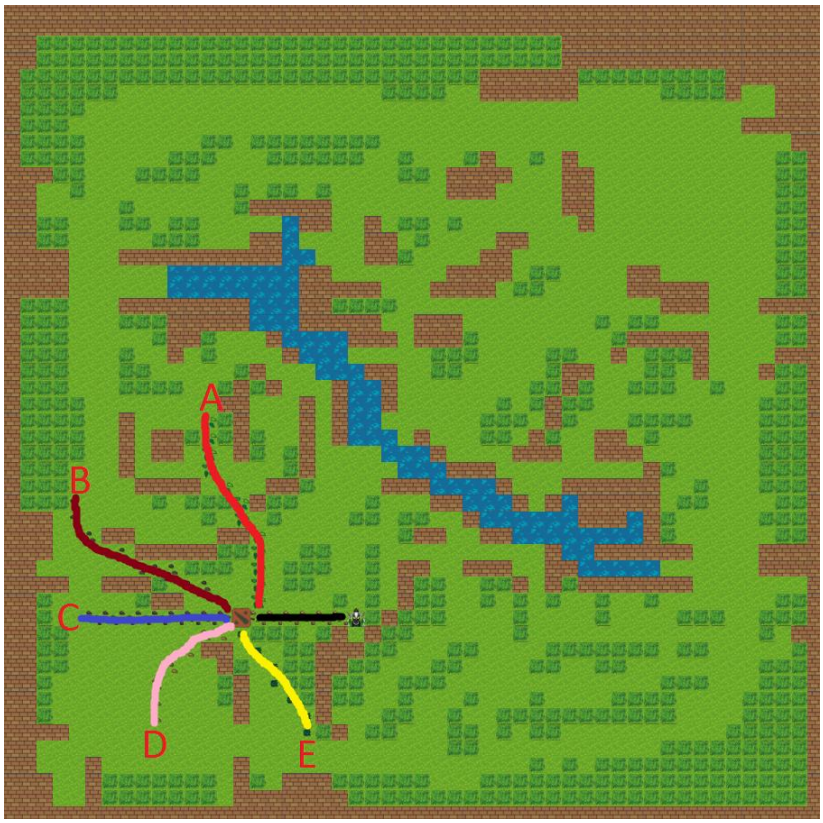

Figura 19: Ferramenta mostrando 6 diferentes caminhos ao mesmo tempo. O normal e outros 5 gravados temporariamente, sendo eles nomeados de A até E.

Nesse caso, são utilizados sprites diferentes para que o usuário entenda e consiga diferenciar sem muitos problemas. Para melhor entendimento neste trabalho, os caminhos foram sobrescritos por linhas coloridas, e iremos nomear cada caminho com uma letra de acordo com a figura anterior.

5.4 Testes Realizados

Inicialmente, foi buscado confirmar o mais simples, ou seja, o caminho mais curto. Foi testado na Figura 20 as mesmas posições para personagens com características bem diferentes.

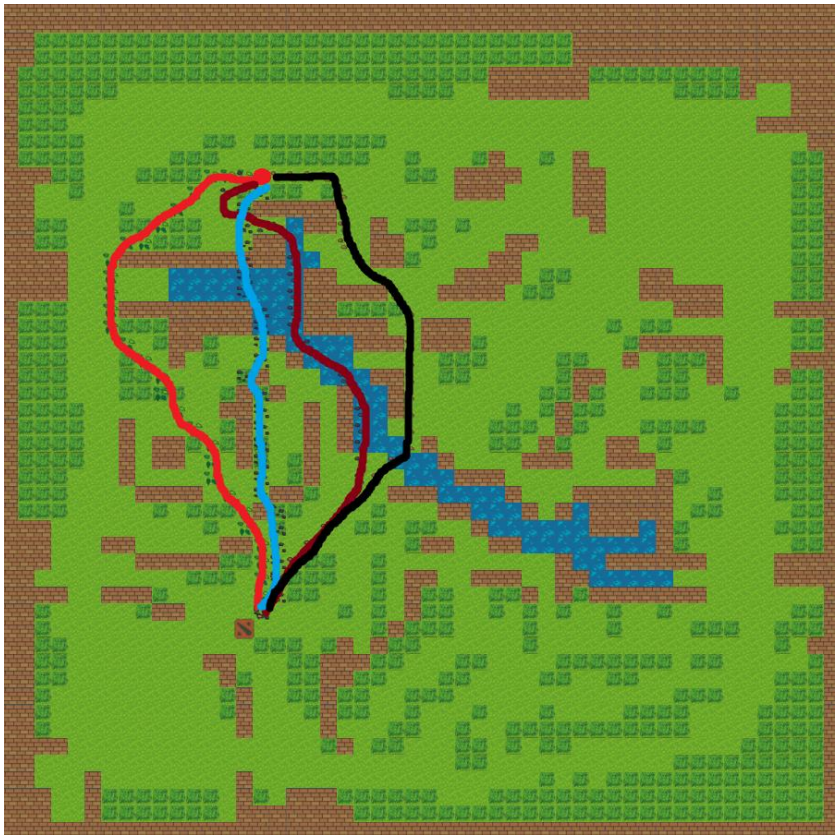

Figura 20: Teste com diferentes personagens na mesma posição.

Para compreender os diferentes caminhos nos testes deste tópico, será utilizada a nomenclatura dada no tópico anterior para cada um.

O primeiro caminho verificado é o A e se refere ao Rubick, personagem que possui mais facilidade de andar na grama, que possui capacidade de saltar sobre as paredes, mas com um custo bem maior. Em seguida foi testado com o Sladar. Verificando suas características e

sabendo que ele tem mais facilidade na água, é fácil perceber que se trata do caminho B. Para o caminho C, quase em linha reta, encontra-se o Viper, personagem voador que pode passar por cima das paredes com um custo bem menor que os outros personagens, mas ainda assim é mais fácil voar pela grama ou água. Por fim encontra-se a Crystal, no caminho D. Ela possui características de terreno bem parecidas com a do Rubick, porém, ela tem um pouco mais de facilidade na água. Contudo, essa pequena diferença foi o suficiente para escolher um outro caminho bem diferente.

No próximo teste, foi considerado que o usuário está com o personagem Rubick e há uma Crystal no caminho. Lembrando que a Crystal possui o menor poder entre os personagens disponíveis, ou seja, é a que menos assusta. Primeiro foi utilizada a estratégia "Muito Defensivo", no qual o personagem decidiu ir por um caminho um pouco mais longo, mas que fosse mais seguro. Depois foi escolhida a estratégia "Defensivo", que neste caso, o personagem escolheu passar pelo caminho mais curto, mesmo tendo um adversário no caminho, considerando que o adversário é fraco. Esse teste é reproduzido na Figura 21.

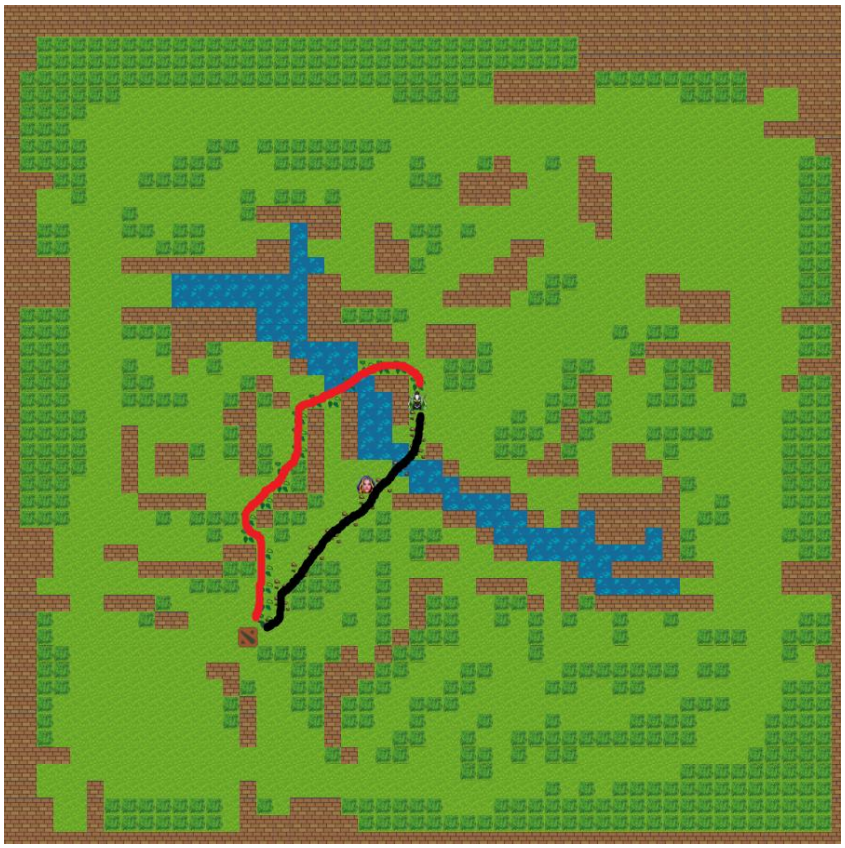

Figura 21: Teste com o mesmo personagem no mesmo local e com diferentes estratégias.

Em uma situação parecida, com o jogador e seu inimigo nos mesmos locais, pode haver mudança de solução caso os inimigos forem de outros tipos. Como no exemplo a seguir, em que a Crystal foi substituída por um Sven, que possui um poder bem maior. Nota-se que o personagem escolheu um caminho mais longo, porém mais seguro em ambas as estratégias, tanto na “Defensivo” como na “Muito Defensivo” como mostra a Figura 22.

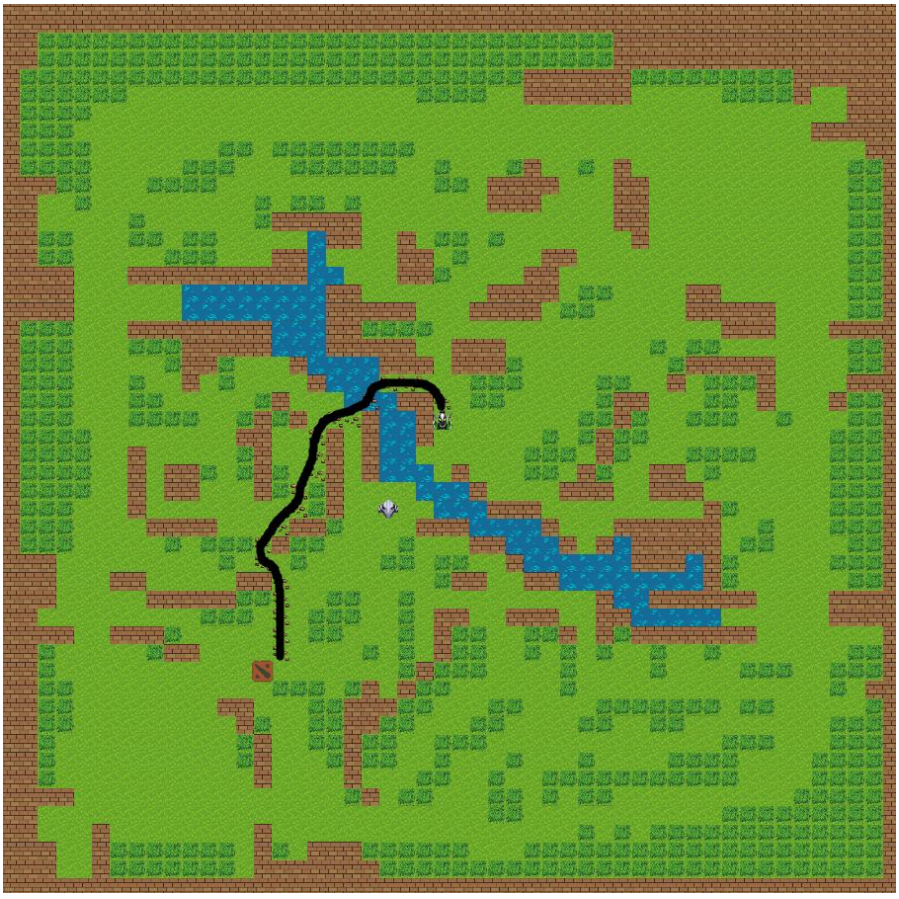

Figura 22: Rubick com estratégia defensiva e um Sven no meio do caminho mais curto.

Além de avaliar o poder dos inimigos, a função também considera bastante a posição do personagem e o custo do caminho desviado, para verificar se vale mesmo escolher o caminho alternativo. Na Figura 23, foi utilizada a estratégia “Muito Defensiva. Percebe-se que no lado direito dessa figura, o personagem escolheu o caminho que passa próximo ao inimigo, pois não valeria a pena atravessar um caminho tão longo pela segurança.

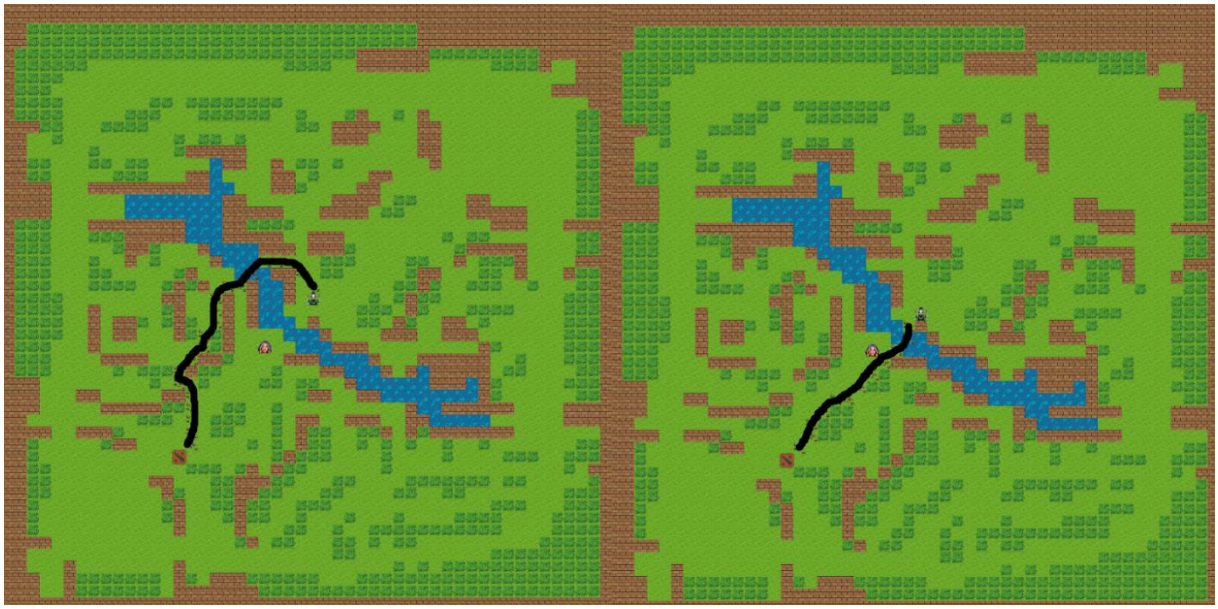

Figura 23: Testes abordando a estratégia muito defensiva, mas em posições diferentes.

Vale lembrar que isso se deve ao grau de defensiva que foi escolhido. Caso o usuário deseje uma defensiva ainda maior, é facilmente implementável, precisando apenas criar uma outra opção com um peso maior para a presença do inimigo. Uma curiosidade para o teste anterior é que se for retirada a possibilidade do personagem se locomover na diagonal, o caminho escolhido irá mudar para um bastante diferente das soluções encontradas até o momento, como mostra na Figura 24.

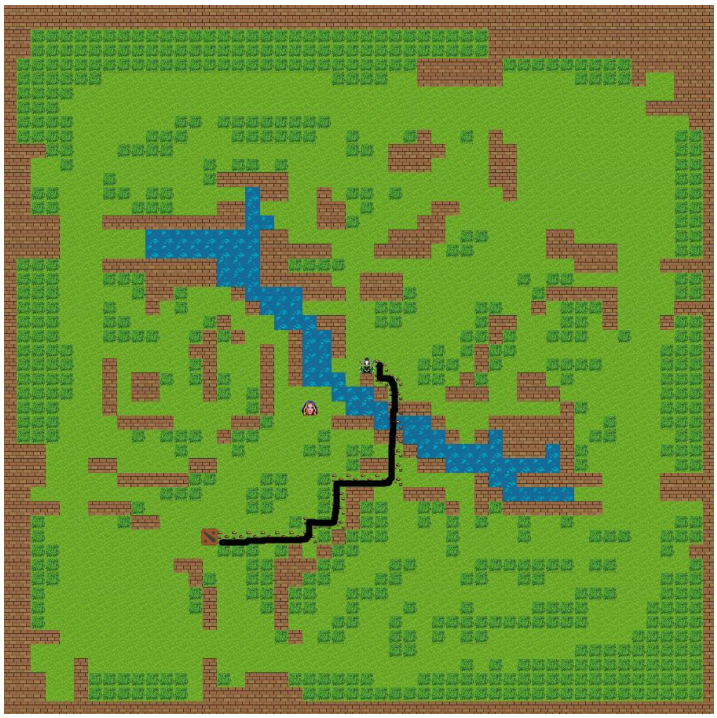

Figura 24: Teste com estratégia muito defensiva sem movimentações na diagonal.

Como já explicado, o usuário também pode escolher considerar o relevo ou não, para calcular o custo do caminho. No exemplo da Figura 25, foi testada a diferença entre o uso ou não do relevo no cálculo. No caminho da direita não está considerando relevo, enquanto no caminho A da esquerda, está.

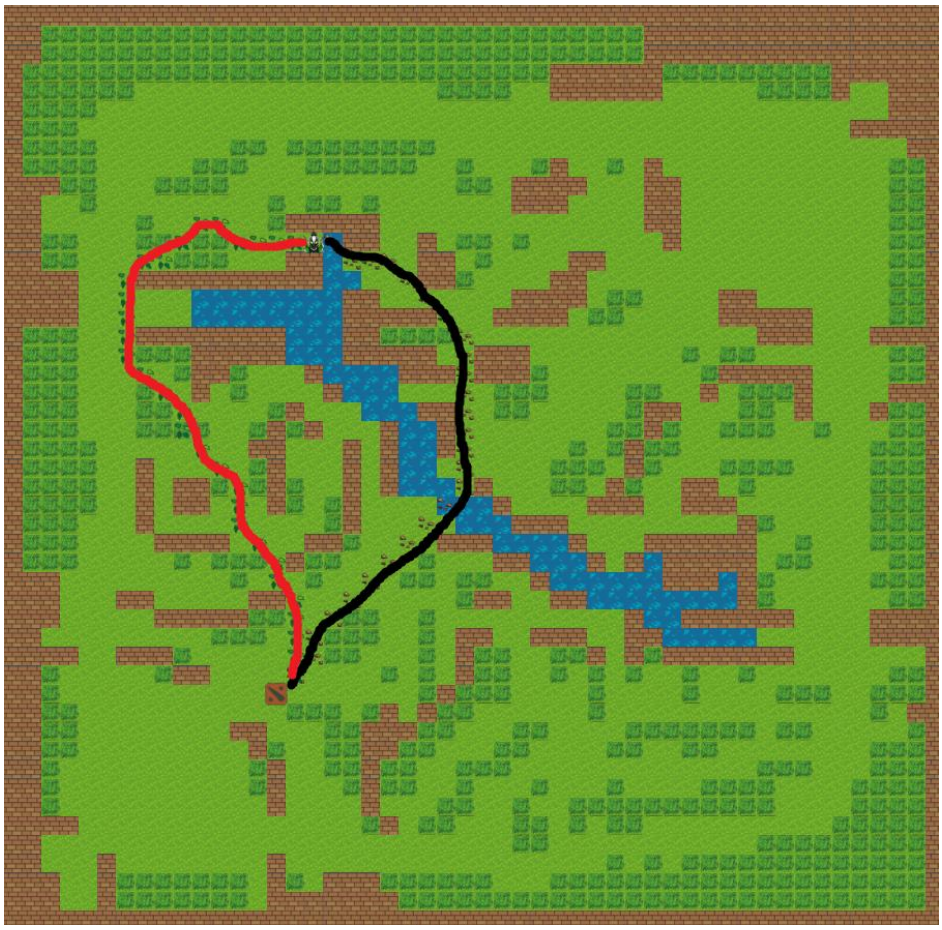

Figura 25: Testes do mesmo personagem na mesma posição, mas considerando ou não relevo.

Esses foram apenas alguns primeiros testes para validar algumas funcionalidades que foram implementadas no modelo desenvolvido. Porém, um grande número de possibilidades podem ser testados, e como já foi citado neste documento, também podem ser adicionadas novas opções no modelo, aumentando assim a quantidade de testes possíveis.

6 CONSIDERAÇÕES FINAIS

Após diversos testes realizados, é possível perceber que encontrar o melhor caminho nem sempre é trivial e nem sempre é simples para o cérebro humano. Considerando as diversidades que o cenário pode oferecer, muitas vezes o caminho pode ser alterado em tempo real. Muitas vezes podem surgir novos adversários durante a execução da movimentação, assim como inimigos podem ser eliminados, ou até mesmo alteração no cenário, como a inclusão ou remoção de barreiras.

O modelo desenvolvido neste trabalho pode ajudar bastante o trabalho dos designers de personagens, pois o permite verificar o comportamento dos personagens com suas preferências em um determinado cenário. O modelo está apto a receber novos parâmetros para serem utilizados no cálculo da função heurística de acordo com a necessidade do usuário.

O simulador desenvolvido foi bastante importante para a representação da execução dos cálculos desejados. Fazendo com que seja possível ver e entender os resultados obtidos com facilidade. É importante lembrar que o simulador foi apenas uma pequena demonstração para um cenário específico, mas que facilmente pode ser alterado para outros estilos de jogos, outros mapas, outros tipos de terrenos e outros personagens.

Enfim, os objetivos propostos no início do trabalho foram alcançados. Foi possível experimentar na prática a junção de vários parâmetros que influenciam no cálculo do *pathfinding* simultaneamente através do modelo criado.

6.1 Trabalhos Futuros

Este trabalho abre as portas para um enorme campo de pesquisa, permitindo que muitas outras características possam ser adicionadas sem precisar alterar a essência do algoritmo principal.

Como possíveis trabalhos futuros, pode-se apontar:

- Desenvolver uma interface mais completa e até mais intuitiva para usuários mais leigos.
- Implementar um leitor de terreno, para que o usuário forneça uma imagem do mapa real que deseja simular e a ferramenta retorne a matriz correspondente.
- Detalhar mais o mapa para aumentar o realismo em relação ao jogo, servindo até mesmo para estudo de jogadores profissionais avaliarem suas jogadas.

- Assim como é considerado um algoritmo de visibilidade no cálculo do melhor caminho neste trabalho, também é um trabalho futuro implementar um algoritmo de detecção de sons, que pode influenciar também na decisão de melhor caminho.
- Trabalhar com multiagentes, ou seja, calcular o melhor caminho considerando vários personagens, podendo tomar decisões diferentes para uma melhor estratégia.

Referências

Red Blob Games. Disponível em : <http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html>. Acesso em 06 de dezembro de 2015

Red Blob Games. Disponível em : <http://www.redblobgames.com/articles/visibility/>. Acesso em 06 de dezembro de 2015

Red Blob Games. Disponível em : <http://www.redblobgames.com/pathfinding/grids/graphs.html>. Acesso em 06 de dezembro de 2015

Algfoor, Z., Sunar, M., & Kolivand, H. (2015). A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *International Journal of Computer Games Technology*.

BOURG, David M.; SEEMANN, Glenn. **AI for game developers**. " O'Reilly Media, Inc.", 2004.

Cormen, T., Leiserson, C., & Rivest, R. (2009). *Introduction to Algorithms, Third Edition.* The MIT Press.

CORRUBLE, Vincent; MADEIRA, Charles AG; RAMALHO, Geber. Steps toward Building of a Good AI for Complex Wargame-Type Simulation Games. In: **GAME-ON**. 2002.

CUI, Xiao; SHI, Hao. A*-based pathfinding in modern computer games. **International Journal of Computer Science and Network Security**, v. 11, n. 1, p. 125-130, 2011.

DECHTER, Rina; PEARL, Judea. Generalized best-first search strategies and the optimality of A*. **Journal of the ACM (JACM)**, v. 32, n. 3, p. 505-536, 1985.

MENDES, Cláudio Lúcio. **Jogos eletrônicos: diversão, poder e subjetivação**. Papirus Editora, 2006.

ONTANÓN, Santiago et al. A survey of real-time strategy game ai research and competition in starcraft. **Computational Intelligence and AI in Games, IEEE Transactions on**, v. 5, n. 4, p. 293-311, 2013.

RABIN, Steve. **AI game programming wisdom**. Cengage Learning, 2002.

TAYLOR, T. L. **Computer Games as Professional Sport: A BIT of Raising the Stakes**. MIT Press, 2014.

TENENBAUM, A. M.; LANGSAM, Y. AUGESTEIN. MJ Estrutura de dados usando C. 1995.