DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA JOGOS MOBA

VICTOR DO NASCIMENTO SILVA

DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA JOGOS MOBA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Chaimowicz

Belo Horizonte

Julho de 2016

Silva, Victor do Nascimento
S586d Desenvolvimento de Agentes Inteligentes para Jogos MOBA / Victor do Nascimento Silva. — Belo Horizonte, 2016
xxv, 85 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Luiz Chaimowicz

1. Computação — Teses 2. Inteligência Artificial — Teses 3. Jogos Eletrônicos — Teses. 4. MOBA. I. Orientador. II. Doutor.

519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Desenvolvimento de agentes inteligentes para jogos MOBA

VICTOR DO NASCIMENTO SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG

PROFA. ROSILANE RIBEIRO DA MOTA
Departamento de Ciência da Computação - PUCMG

Belo Horizonte, 14 de julho de 2016.

Para meus irmãos:
Raissa, a Justa.
Emanuella, a Destemida.
João, o Gentil.
Natã, o Magnífico.

E para a minha vó da ciência, Therezinha Guimarães (in memorian).

Agradecimentos

Agradeço aos meus irmãos: Raissa, Emanuella, João e Natã. Que por vezes ficavam bravos porque eu não queria parar para lhes dar atenção enquanto estudava. Obrigado pela algazarra e pela curiosidade.

Aos mestres e amigos: João Paulo, Marcelo Nery, Michelle Nery e Rosilane Mota, vocês foram parte das pessoas que me abraçaram incondicionalmente, obrigado pela dedicação. Obrigado também a Aline Silva, Letícia Lopes e o Diego Marinho. Vocês foram minhas inseparáveis companhias, sempre me jogando pra cima.

Nunca fui de muitos amigos, mas sou grato à vida por ter me dado uma amiga que estava ali do meu lado para tudo. Juntos festejamos, choramos, rimos, brincamos, adoecemos, no curamos e continuamos aqui, firmes e fortes. Dayane, um obrigado não é suficiente e você sabe disso!

A reta final foi especialmente desafiadora, e tenho certeza que não estaria aqui agradecendo sem o apoio de pessoas que me ajudaram além do imaginável. Obrigado à Elizabeth Etevalde e à Therezinha Ethevalde, que me acolheram e tornaram possível a continuidade desse trabalho. Vocês são minha família de coração!

Acredito que as principais viagens não são físicas, por isso é importante lembrar das pessoas que estiveram viajando comigo em pensamentos. Obrigado pessoal do Laboratório J: Mirna Silva, Fabrício Rodrigues, Yuri Macedo e Gianlucca Zuin.

Thanks to my remote friends Jay Rosenberg and Hendrik Schulze, that helped me during those years in many ways. I had such fun times working in projects with you guys. Hope we can work together again. GG!

Devo grande parte destas conquistas também ao meu orientador: obrigado *Chaimo*, por ter aceitado me guiar nessa jornada e confiado em mim. Obrigado também por aprender um tanto sobre MOBAs, heróis, magias e meu linguajar confuso para que pudéssemos executar esse trabalho. Ele não seria nada sem você.

Obrigado aos funcionários do PPGCC, a FAPEMIG e a CAPES. Obrigado a todos que me apoiaram, direta e indiretamente. Com um pouquinho de cada e uma multidão de pessoas, vocês fizeram toda a diferença.

"Até mesmo uma boa decisão, se tomada pelos motivos errados, pode ser uma má decisão."

(Governador Weatherby Swann, Piratas do Caribe)

Resumo

MOBA (do inglês, *Multiplayer Online Battle Arena*) é um dos gêneros de jogo mais jogados no mundo atualmente. Apesar da grande popularidade dos MOBAs entre a comunidade de jogadores, este recebe pouca atenção da comunidade acadêmica. Com seu constante crescimento, se faz necessário o desenvolvimento de Agentes Inteligentes que sejam capazes de jogar juntamente e contra jogadores humanos. Estes agentes podem ter uma vasta gama de utilidades, como o auxílio aos novos jogadores ou o treinamento de jogadores profissionais. Neste trabalho, procura-se propor uma solução para o problema de desenvolvimento de agentes inteligentes para jogos MOBA. São apresentados dois agentes: um agente baseado numa arquitetura heterogênea de análise tática capaz de jogar MOBA; e um agente de suporte aos novos jogadores. Até o desenvolvimento desta pesquisa, não haviam grandes avanços no desenvolvimento de agentes inteligentes para jogos MOBA. Até onde se sabe, há apenas uma solução proposta baseada na técnica de Aprendizado por Reforço, implementada sem grande sucesso. Na solução aqui proposta, utilizou-se a modelagem de um parâmetro denominado *aggro*, que consiste em interpretar a agressividade presente no ambiente. Para o armazenamento e manutenção deste parâmetro, foi utilizada a técnica de Mapa de Influência, sendo o ambiente uniformemente discretizado. Uma arquitetura com módulos independentes de combate e movimentação foi utilizada no controle do agente. Como resultado, obteve-se um agente capaz de tomar decisões no ambiente MOBA. Além disso, o agente foi capaz de realizar a mecânica de *kiting*, que consiste em bater e correr. Anteriormente, esta mecânica exigia módulos dedicados para sua implementação. Para o agente de suporte ao jogador, foram estudadas várias abordagens como Máquina de Estado Finito, Planejamento Baseado e Objetivos e Árvore de Comportamentos. A última técnica foi aplicada juntamente a um Sistema Baseado em Regras para compor um agente capaz de jogar juntamente ao jogador e dar dicas ao mesmo. Os agentes são então submetidos a testes validadores. O agente de jogo para MOBA demonstrou eficácia no ambiente MOBA, apresentando ainda sua eficiência na coleta de recursos. Por sua vez, o agente de suporte foi validado por jogadores como útil e válido no auxílio de novos jogadores

no aprendizado de jogos MOBA.

Palavras-chave: MOBA, Agentes Inteligentes, Inteligência Artificial, Análise Tática, League of Legends.

Abstract

Multiplayer Online Battle Arena (MOBA) is one of the world's most played game genres. Despite its popularity among the gamer community, there is still little attention from the Academia to this specific game genre. With the growth of MOBA games, its necessary to develop intelligent agents capable of playing alongside and competing against human players. These agents can have a wide range of applications, such as helping new players to learn how to player MOBA or training professional level players. In this work, we propose an approach for the development of intelligent agents for MOBA games. We present two agents in our approach: the first one consists of an agent with a heterogeneous architecture that uses tactical analysis for playing MOBA while the second is an agent capable of supporting new players. To the best of our knowledge, this is one of the first works to address the problem of developing agents for MOBA. Our solution models the *aggro* parameter, which consists of a metric to the aggressiveness present in the environment. To do this, our approach uses an Influence Map to store and manage the parameter, using a uniform discretization of the environment. A heterogeneous architecture with decoupled combat and navigation modules is proposed for the agent's decision making. The agent is also capable of emerging the *kiting* behavior, which means that the agent is capable of hit and flee. In other works, this behavior required a dedicated module to be implemented. Moreover, we have researched different techniques for the implementation of the support agent. Among these techniques, we include Finite State Machines, Goal Oriented Action Planning, and Behavior Tree. The Behavior Tree has been used for the development of the support agent with a Rule Based System that was used to develop a player's tip system. Different experiments were performed to validate our approach. The MOBA playing agent shows to be effective in the game environment. Moreover, the agent shows to be efficient in the resource collection task. On the other hand, the support agent is validated by human players, which consider that the agent is valid for the use with new players during their MOBA learning phase.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Neste capítulo são introduzidos os principais tópicos desenvolvidos nesta dissertação. Primeiramente se apresenta as motivações desta pesquisa. Posteriormente, apresenta-se os objetivos e, em seguida, as contribuições realizadas por esta. Finalmente, apresenta-se a organização do restante deste texto.

1.1 Motivação

Os jogos MOBA (do inglês, *Multiplayer Online Battle Arena*) são compreendidos por um gênero de jogo que tem suas origens nos jogos de Estratégia em Tempo Real (do inglês, *Real-Time Strategy* ou simplesmente RTS). Em suas primeiras aparições, os jogos MOBA representaram o desenvolvimento de características que não eram fortemente exploradas em jogos RTS, como o *micromanagement* e a itemização. Além destas características, o gênero MOBA apresentou um estilo de jogabilidade única, o que chamou a atenção de milhões de jogadores ao redor do mundo. Atualmente, MOBA é um dos gêneros mais jogados do mundo, cerca de 30% da jogabilidade *online* mundial [Murphy, 2015].

Apesar de toda a atenção da comunidade mundial de jogadores ao gênero, nota-se uma reduzida aproximação da academia nessa direção em relação a outros gêneros de jogo. Muito provavelmente esta fraca aproximação se deve ao fato de a maioria dos jogos do gênero serem comerciais, o que dificulta a utilização destes jogos como plataforma de testes para pesquisas. Acredita-se que a utilização destes jogos como plataforma de testes para a pesquisa em Inteligência Artificial possa ser benéfica, proporcionando um ambiente controlado, seguro e de baixo custo, tendo claras condições de vitória, derrota e empate [Laird, 2002]. Esta crença se deve ao fato de outros gêneros de jogo já serem utilizados como plataformas de teste com sucesso [Ontanón et al., 2013].

Apesar de o gênero MOBA ter conquistado pouco espaço na academia, nota-se que o seu antecessor, o gênero RTS, é largamente utilizado como plataforma de pesquisa. Para jogos como Starcraft, há competições específicas de agentes e trilhas especiais em conferências, dedicadas ao estudo de agentes que sejam capazes de demonstrar domínio sobre o jogo [Ontanón et al., 2013]. Embora jogos MOBA tenham suas raízes em jogos RTS, o foco de pesquisa e os desafios encontrados nestes gêneros é bem distinta. A pesquisa em RTS foca na administração de recurso e controle de grandes massas de unidades, habilidade denominada *macromanagement*. Por outro lado, a pesquisa em MOBA foca nas minúcias mecânicas, controle fino e demonstração de domínio de uma unidade específica, habilidade chamada de *micromanagement*. Outra diferença marcante é que enquanto RTS é comumente jogado por um jogador contra outro, o gênero MOBA permite que até cinco jogadores se unam em um mesmo time, totalizando 10 jogadores por partida. Assim, enquanto na maioria dos casos as partidas RTS são jogadas individualmente, as partidas de MOBA são normalmente jogadas por múltiplos jogadores. Esta diferenciação exige um conjunto de tomada de decisão diferente, uma vez que é requerida a cooperação entre os jogadores em uma partida de MOBA, fato que não acontece em uma partida de RTS.

Executar uma mecânica eficiente utilizando uma unidade dentro de um jogo MOBA, requer a análise de uma grande quantidade de variáveis. Estas mecânicas, as quais denomina-se *micromanagement*, foram estudadas inicialmente no contexto de RTS [Uriarte & Ontañón, 2012], executando o isolamento de unidades para a simulação de cenários de *micromanagement*. Para tanto, foi necessária a coleta de dados sobre o ambiente, unidades, habilidades, desempenho, dentre outras, formando um cenário complexo a ser analisado. A técnica de isolamento e análise de unidades e ambiente foi explorada por Hagelbäck & Johansson [2008b] e Uriarte & Ontañón [2012] utilizando o jogo Starcraft: Brood War como plataforma de teste. Acredita-se que um melhor desempenho em *micromanagement* pode ser atingida uma vez que uma análise mais específica do cenário e suas variáveis seja realizada, e que um ambiente propício seja disponibilizado. O ambiente MOBA oferece um cenário favorável para o desenvolvimento de análises de *micromanagement*, proporcionando dados e cenário que podem ser utilizados para o desenvolvimento de agentes especialistas sem a modificação ou isolamento de características do jogo.

Como motivação secundária, lista-se a curva de aprendizado encontrada em jogos MOBA, tornando jogos deste gênero complexos tanto para agentes quanto para humanos. Outro fator encontrado no gênero MOBA é a interação com outros jogadores inerente ao processo de jogar. Muitas das vezes, estas interações entre jogadores não são amigáveis, gerando as chamadas interações *tóxicas* [Kwak et al., 2015]. Jogadores

tóxicos são aqueles que interagem de forma negativa com outros jogadores, utilizando xingamentos, desestímulos ou atrapalhando outros jogadores. O comportamento tóxico é um problema encontrado frequentemente em jogos online, e enquadra-se no chamado *cyberbullying*. Este comportamento, juntamente à curva de aprendizado encontrada no gênero MOBA, pode vir a desestimular jogadores iniciantes a continuar a jogar. Portanto, acredita-se que é necessária a minimização do comportamento tóxico e da curva de aprendizado para que o jogador possa atingir um alto nível de imersão durante as partidas.

1.2 Objetivo

O principal objetivo deste trabalho é propor um agente inteligente que seja capaz de jogar partidas de jogos do gênero MOBA. Para tanto, explora-se diferentes técnicas a fim de propor um agente que seja capaz de se comportar de forma racional dentro de um contexto de jogo do gênero MOBA.

A questão principal a ser respondida neste trabalho é: *Dado um jogo MOBA, é possível ter um agente que possa executar as mecânicas básicas de um personagem qualquer e vencer o jogo?*

Como objetivo secundário, propõe-se o desenvolvimento de um agente inteligente que possa auxiliar novos jogadores em suas partidas iniciais. Este agente deve ser capaz de jogar juntamente ao jogador, evitando o contato inicial com jogadores tóxicos e mitigando o desestímulo do jogador. Além de evitar o contato com jogadores que possam eventualmente desestimular o jogador iniciante, estes agente tem o papel principal de tutor. Portanto, o agente deverá ser capaz de ensinar o jogador, dando dicas sobre passos iniciais na jogabilidade MOBA. Além disso, deverá ser capaz de analisar e indicar deficiências no desempenho do jogador durante ou em fases específicas de uma partida.

Este trabalho propõe ainda popularizar o desenvolvimento de agentes inteligentes para jogos do gênero MOBA e o uso deste gênero como plataforma de pesquisa. Esta proposição se torna importante uma vez que a academia tem dado muita atenção aos jogos do gênero RTS, e o fato de o gênero MOBA compartilham algumas características com o gênero RTS, podendo ser de grande interesse. Mais ainda, o gênero MOBA é atualmente o gênero de jogo mais jogado no mundo, com uma grande base de jogadores que consome mais de 30% da jogabilidade online [Murphy, 2015].

1.3 Contribuições

Como principais contribuições deste trabalho lista-se:

- uma arquitetura baseada em análise tática como principal mecanismo de tomada de decisão de um agente inteligente em jogos MOBA;
- o desenvolvimento de um agente inteligente que é capaz de jogar MOBA utilizando uma arquitetura multi-camada;
- o desenvolvimento de um agente inteligente capaz de auxiliar novos jogadores a aprenderem a jogar League of Legends;
- o uso de jogos do gênero MOBA como plataforma de testes para pesquisa em IA, em particular, League of Legends;
- um panorama dos principais desafios de IA relacionados aos jogos MOBA.

Estas contribuições são apresentadas em três artigos:

- em [Silva & Chaimowicz, 2015a], apresentou-se um agente baseado numa arquitetura multi-camadas capaz de jogar uma partida de MOBA efetivamente. Mais ainda, o agente foi capaz de emergir o comportamento de *kiting*[1], demonstrado anteriormente na literatura apenas com módulos dedicados;
- apresentou-se também um agente que foi capaz de auxiliar novos jogadores em [Silva & Chaimowicz, 2015b]. Discutiu-se ainda as características e benefícios da utilização deste agente, realizando testes com usuários;
- com o conhecimento e características exploradas em diversos jogos MOBA, publicou-se [Silva et al., 2015] onde se discute o balanceamento dinâmico de dificuldade em jogos MOBA, utilizando o jogo DotA como plataforma de testes.

Finalmente, além dos trabalhos já publicados, encontra-se submetido o artigo [Silva et al., 2016], onde estende-se o trabalho desenvolvido em [Silva et al., 2015]. Nesse trabalho, executou-se ainda testes com usuários e uma maior exploração do gênero MOBA.

[1] Kiting é o ato de atacar e correr em jogos de estratégia.

1.4 Organização

Este trabalho está organizado em seis capítulos, sendo estes descritos a seguir:

Capítulo 2 Discute as características e jogabilidade de jogos MOBA. Discute ainda os trabalhos relacionados, abordando os temas necessários ao desenvolvimento deste trabalho e a literatura sobre MOBA disponível;

Capítulo 3 Discute as técnicas utilizadas neste trabalho, discorrendo sobre características e implementação. Discorre ainda sobre o processo de seleção de plataforma de pesquisa para este trabalho;

Capítulo 4 Discute o desenvolvimento dos dois agentes implementados neste trabalho. Primeiramente sobre o agente de jogo para o ambiente MOBA e então um agente de suporte ao jogador iniciante.

Capítulo 5 Compreende os experimentos realizados para a validação dos agentes desenvolvidos.

Capítulo 6 Finaliza este trabalho com as conclusões e trabalhos futuros.

Capítulo 2

Jogos MOBA e Trabalhos Relacionados

Neste capítulo é apresentado o gênero de jogo MOBA, o qual é o objeto de estudo deste trabalho. A compreensão da plataforma se mostra necessária para o entendimento do conhecimento aqui apresentado, por isso dedica-se especial atenção à mesma. Apresenta-se ainda o jogo League of Legends, plataforma de testes escolhida para a representação do gênero nos experimentos realizados.

Posteriormente, discute-se os trabalhos relacionados, englobando conhecimento sobre MOBA ou RTS, gênero antecessor e que compartilha características com o gênero aqui estudado.

2.1 Multiplayer Online Battle Arena

Os jogos MOBA tem suas origens nos jogos do gênero RTS, muito populares em meados dos anos 90 e início dos anos 2000. Suas características herdam traços que estão presentes em ambos os gêneros, sendo que os primeiros MOBAs foram criados utilizando as próprias ferramentas de criação de mapas RTS. Para que se possa entender melhor as origens e características dos jogos MOBA, apresenta-se nesta seção um breve histórico de evolução do gênero.

Com a popularidade dos jogos de estratégia, houve grande interesse por parte dos jogadores em criar modos de jogo e mapas personalizados para estes. Além disso, juntamente a estes jogos, eram fornecidas ferramentas de criação de mapas personalizados. Com esta disponibilidade os usuários se tornaram modificadores do conteúdo dos jogos, sendo denominados *modders*. Os *modders* eram responsáveis pelo desenvolvimento de mapas alternativos, e muitos destes tendiam para o estilo de jogo de

um MOBA. Desde o início do desenvolvimento do gênero, a jogabilidade consistia em controlar uma unidade poderosa denominada "herói"ou "campeão", escolher entre três rotas para avançar suas tropas e conquistar uma estrutura principal inimiga.

Um dos primeiros MOBAs de sucesso teve origem no jogo da produtora *Blizzard*, o RTS Starcraft, sendo um mapa modificado do mesmo que ficou conhecido como AeoN of Strife. Este sendo um dos pioneiros no estilo de jogabilidade, e também à fama de Starcraft, foi muito bem recebido pela comunidade. Após o sucesso de AeoN of Strife, houve outro jogo que chamou a atenção da comunidade *modder*: o também RTS da *Blizzard*, Warcraft III. Este jogo era essencialmente parecido com Starcraft, porém tinha raças diferentes, gráficos tridimensionais melhores, uma gama de cenários novos e ambiente de personalização mais poderoso, atrelado inclusive a uma linguagem de programação. A comunidade *modder* voltou então a sua atenção para Warcraft III, criando uma grande quantidade de mapas com jogabilidade parecida com os MOBAs de hoje. Nesse momento, surgiu um dos mapas mais populares de todos os tempos, o Defense of the Ancients, ou simplesmente DotA. O sucesso foi tão grande que muitos jogadores passaram a se dedicar exclusivamente a este mapa, o que permitiu a criação de um cenário específico para o DotA na comunidade de jogadores. As produtoras inclusive passaram a investir em campeonatos e competições específicas para o DotA. O sucesso foi tamanho que houve a criação de plataformas competitivas, como a *Garena*, que passaram a dar suporte específico para partidas de DotA.

Devido ao sucesso de DotA, algumas desenvolvedoras resolveram apostar no estilo de jogo, sendo que em 2009 a *Riot Games* lançou o seu jogo League of Legends (LoL). Este, até os dias atuais, é o jogo mais jogado do mundo, tendo superado o jogo World of Warcraft em meados de 2012 [Gaudiosi, 2012]. Até então o termo MOBA era inexistente e os jogos eram simplesmente denominados "Aeon like", "DotA like"e "Action Real-Time Strategy". A *Riot Games* foi considerada a responsável por cunhar o termo "Multiplayer Online Battle Arena".

Além da *Riot Games*, a *Valve* resolveu apostar no mercado MOBA, lançando uma nova versão do jogo Dota, o jogo Dota2. Neste, os mesmos personagens e jogabilidades de DotA foram mantidos, porém recriados em uma plataforma dedicada e com gráficos melhorados. Outros jogos originados de DotA também ganharam forma, como o Heroes of Newerth, onde a jogabilidade de DotA foi reproduzida com personagens diferentes. Outros jogos de destaque na atualidade do gênero MOBA são: Heroes of the Storm, da *Blizzard*; Strife, da *S2 games*; e Smite, da *LevelUp!*.

2.2 Jogabilidade MOBA

Em geral, jogos MOBA tem características de jogo muito singulares se comparadas a outros gêneros. Porém, quando comparados entre si, os jogos MOBA se mostram bastante similares em jogabilidade e características. Nesta seção discutir-se-á estas características indicando as semelhanças e similaridades, quando existentes.

O MOBA é um jogo que exige planejamento e rápida tomada de decisão, a curto e a longo prazo. Se caracteriza ainda como um jogo em tempo real e com ambiente parcialmente observável. Devido às probabilidades presentes em ataques críticos e em alguns poderes dos heróis, jogos do gênero MOBA são classificados como não determinísticos. A cada partida o jogador pode escolher entre um dos personagens disponíveis, e deve desenvolvê-lo ao longo do jogo. Toda vez que uma nova partida for iniciada, o herói retorna ao nível inicial e o mundo de jogo volta ao estado inicial. Devido a estas características, jogos MOBA são classificados como não persistentes.

O jogo é normalmente jogado por grupos de 10 jogadores, sendo cinco em cada time. Cada jogador deverá então selecionar um herói com o qual prosseguirá jogando pelo resto da partida. Este herói é uma unidade mais poderosa que as demais, dotada de habilidades específicas, características e *status* que tornam sua jogabilidade única dentro de uma partida.

Ao iniciar o jogo cada jogador recebe uma quantia de dinheiro, que deve ser utilizada para adquirir itens a fim de melhorar seu *status* ou obter habilidades especiais. O jogador recebe ainda quantidades de dinheiro progressivamente ao longo do jogo. Após determinado tempo, algumas unidades controladas pela inteligência artificial do jogo são criadas e seguem por uma das três rotas do mapa, como visto nas partes em amarelo da Figura 2.1. Normalmente estas unidades são criadas em grupos, que surgem em intervalos determinados de tempo. Estas unidades controladas pela IA são denominadas *creeps*, *minions* ou *tropas*. Ao abater as tropas inimigas, o jogador pode obter mais dinheiro. Para tanto deve executar uma ação denominada *last-hit*, que consiste em desferir o último ataque àquela unidade. O jogador pode ainda obter mais dinheiro abatendo heróis inimigos ou unidades neutras que habitam o mapa de jogo. Tais unidades neutras não são agressivas, e se encontram em pontos específicos entre as rotas, numa área denominada *selva*. Um mapa generalizado do jogo é apresentado na Figura 2.1.

Para vencer o jogo, o jogador deve utilizar seu herói para, juntamente com seus companheiros, derrotar uma série de estruturas fortificadas presentes em uma ou mais rotas e capturar a estrutura principal do time inimigo. O jogador encontrará na rota, além das tropas, estruturas defensivas, como as torres. É possível ainda encontrar

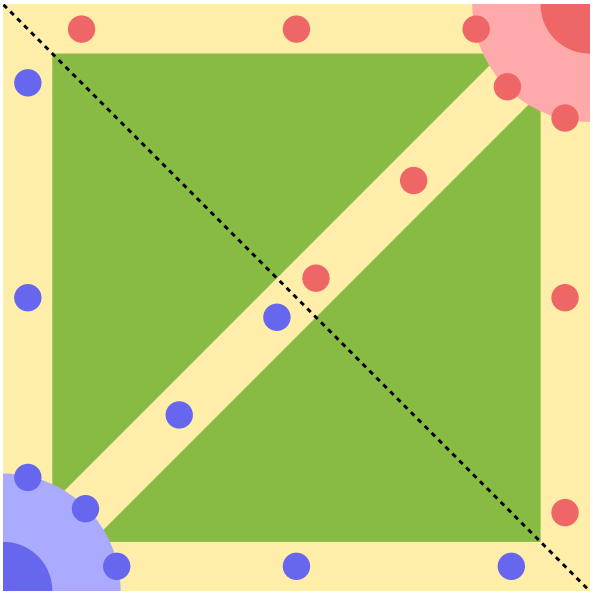

Figura 2.1. Mapa generalizado de um jogo MOBA, normalmente composto por três rotas. A área em amarelo mostra as rotas do mapa. Os grandes semi-círculos em vermelho e azul mostram as bases do time. A linha tracejada mostra a divisão do mapa, uma área comumente chamada de rio. Os pontos em azul ou vermelho ao longo das rotas representam as torres.

estruturas que, se capturadas, concedem vantagens ao time, como tropas mais fortes.

O jogador pode ainda ser punido por suas más decisões ou falhas mecânicas durante a jogabilidade. A pior destas punições é a "morte"do herói, que acontece quando o HP (do inglês, *Health Points*, ou "pontos de saúde") do personagem é zerado. O personagem então recebe uma punição, em que deverá continuar morto por um determinado período de tempo. Durante o estado de morte, o personagem não pode realizar nenhuma ação. Outras punições possíveis são: silenciar, consistindo em não poder usar habilidades; atordoar, não podendo realizar ações; enraizar, não podendo se mover; e desacelerar, tendo a mobilidade debilitada.

Em essência, uma partida de jogo MOBA é bastante similar a uma partida do jogo popular denominado "rouba bandeiras". Um grupo de jogadores deverá derrotar uma série de jogadores adversários e capturar a estrutura principal do time inimigo. Caso seja pego e derrotado no caminho, o jogador recebe uma punição de tempo em que não pode realizar ações. Vence o time que conquistar a estrutura principal do inimigo primeiro.

Utiliza-se nesta pesquisa o jogo League of Legends como plataforma para realizar os testes propostos. Essa plataforma foi selecionada baseada em critérios técnicos que permitissem à pesquisa demonstrar claramente as características únicas de jogabilidade,

desenvolvimento de agentes e tutores digitais em jogos MOBA. É importante ressaltar que tais características devem ser levadas em conta sempre que a pesquisa utilize jogos como plataforma de testes, devido à restrição oferecida por estes domínios.

MOBAs possuem uma natureza competitiva inerente onde dois times de jogadores se opõem a fim de capturar a estrutura principal do time inimigo. No modo de jogo mais comum, cada time é composto por cinco jogadores. Cada jogador é representado por seu respectivo herói, que deve ser selecionado antes do início da partida, a partir de uma grande variedade de heróis disponíveis. Cada herói possui características e habilidades únicas, que podem ser utilizadas para assumir um papel dentro da partida. Devido à característica não persistente do jogo, a cada partida o jogador recebe um herói no nível 1, devendo desenvolver esse herói para que chegue ao nível máximo permitido pelo jogo (em League of Legends os níveis vão até 18). O jogador deve manipular o herói selecionado para lutar com inimigos, tropas e unidades neutras. Ao derrotar estas unidades o jogador recebe dinheiro, que pode ser utilizado para adquirir itens. Além disso, o jogador recebe pontos de experiência, que aumentam o nível do herói. A vitória em League of Legends é atingida quando um time captura a estrutura principal na base inimiga, denominada *Nexus*.

Ao escolher um herói, o jogador deve compreender as mecânicas básicas daquele herói e quais os papéis mais comumente ocupados por ele. Há heróis que podem ser utilizados em várias funções, mas a maioria destes será muito melhor aproveitada caso assuma a função para o qual foi criado. Em LoL existem algumas funções básicas:

- *Tanque*: é o herói que tem a capacidade de receber grandes quantidades de dano sem ser abatido;

- *Assassino*: consiste em um herói que tem a capacidade de causar grandes quantidades de dano a um alvo único, podendo finalizar um campeão com facilidade;

- *Mago*: É o herói que é capaz de infringir dano utilizando principalmente suas habilidades. Normalmente este herói infringe grandes quantidades de dano mágico, sendo pouco eficiente em causar dano físico;

- *Suporte*: é o herói que tem por principal função auxiliar o time com suas habilidades. Normalmente estes campeões possuem habilidades que podem aumentar os *status* dos aliados ou diminuir os *status* dos campeões inimigos. Além disso, comumente possuem habilidades de controle de grupo, responsáveis por punir heróis do time adversário;

- *Atirador*: herói que possui grande capacidade de causar dano à distância. Normalmente é auxiliado por um suporte, pois pode ser facilmente eliminado. Estes heróis podem causar grandes quantidades de dano físico, mas normalmente não causam grandes quantidades de dano mágico;

- *Lutador*: herói que possui como principal característica a entrada fisicamente em lutas. Estes heróis normalmente atacam de perto, sendo classificados como *melee*. Podem atuar como iniciadores e normalmente jogam na rota do topo;

- *Jungler*: é o herói que é responsável por evoluir lutando contra tropas neutras na selva aliada. Este herói é ainda responsável por atuar realizando emboscadas, o que o atribui a tarefa de *ganker*. Engloba uma vasta gama de heróis, como magos, assassinos e lutadores.

É importante observar que um mesmo herói pode ter uma ou mais funções. Um mago por exemplo pode atuar na função de suporte e vice-versa. Mais do que isto, a fusão de duas classes pode gerar uma sub-classe. Por exemplo, um herói que atua ao mesmo tempo como *Lutador* e *Tanque* é comumente denominado *Bruiser*, devido a sua grande capacidade de causar e absorver dano.

Em outros jogos, como Dota2 e Smite, há uma diferenciação enorme em relação à nomenclatura e às funções. Todavia se observa que os papéis e as ações que são executadas são, em sua essência, as mesmas. Em Dota2 por exemplo existem comumente as funções de *Ganker* e *Iniciador*, condições que não são primariamente encontradas em LoL, mas que podem ser interpretadas como o *Jungler*, ou o herói da rota do topo, conforme o caso.

Existem outros conceitos que devem ser considerados durante a seleção do herói. Primeiramente, a formação aliada, ou seja, os heróis escolhidos pelos jogadores aliados, aumentando assim a sinergia entre os mesmos e permitindo a execução de combos[1]. Além disso, é necessário analisar um conjunto de informações do jogo e de fora dele para prever quais os heróis, técnicas, formações e abordagens são mais eficientes durante um determinado período de tempo, conjunto este que é comumente denominado *metagame*.

O *metagame* é um conceito utilizado em jogos para descrever o estado-da-arte em estratégia em um determinado jogo. Ele é formado pela própria comunidade e é fortemente influenciado por estratégias apresentadas por jogadores profissionais. Um bom exemplo é o de que se percebe que um herói pode ser utilizado para se jogar

[1]Combos são combinações de habilidade de um ou mais heróis a fim de causar maior dano ou conseguir vantagem sobre o time inimigo.

contra as estratégias adotadas atualmente no jogo. Assim que se percebe esta característica, muitos jogadores tendem a migrar para tal estratégia. Quando se percebe que esta estratégia se tornou comum, é necessária a geração de novas estratégias que sejam eficientes contra o estado-da-arte atual. Além disso, conhecimentos como estratégias comumente adotadas por times, habilidades de jogadores de determinado nível de habilidade e nível de habilidade necessária para a execução de estratégias são fortes influenciadores do *metagame*.

A natureza evolutiva dos MOBAs faz com que o *metagame* mude constantemente, sendo necessária a adaptação e criação de novas estratégias, garantindo assim a manutenção do ecossistema do jogo. Um exemplo disso é a compilação de Heróis mais comum em League of Legends, no qual dois jogadores vão para a rota inferior, sendo um *Suporte* e um *carregador*, na rota do meio pode existir um *Mago* ou um *Assassino*. No topo, joga um *Tanque* ou *Lutador*, geralmente agindo como iniciador das lutas. Por fim, na selva normalmente há um *Lutador*, *Mago*, ou qualquer outro gênero de herói que se adapte à posição, sendo comumente chamado de *Jungler*.

2.3 Trabalhos Relacionados

Jogos MOBA são ainda pouco explorados pela Academia, todavia, nota-se o aumento do interesse nestes jogos nos últimos anos. Parte deste pequeno interesse se deve ao fato de os jogos MOBA, em sua grande maioria, serem comerciais, o que dificulta sua utilização em trabalhos de pesquisa. Mais do que isto, jogos MOBA dependem de um servidor de dados da empresa e são executados *online*, o que torna sua execução dependente de outros recursos. Finalmente, esta execução *online* pode gerar alguns ruídos durante os testes, uma vez que o cliente de jogo está sujeito à queda de conexão, falhas na rede e também atrasos.

O problema de acesso às APIs (do inglês, *Application Programming Interface*, ou Interface de Programação de Aplicação) de jogo também foram enfrentadas pelos jogos RTS durante suas primeiras fases de implementação como plataforma de pesquisa. A forma adotada pelos pesquisadores foi a utilização de ferramentas de terceiros. O caso de maior amplitude é o da utilização do jogo Starcraft como ferramenta de pesquisa. Para tanto, foi necessária a utilização de uma ferramenta chamada BWAPI para a implementação de agentes de jogo [Buro & Churchill, 2012].

Com a maior disponibilidade de ferramentas de pesquisa, os pesquisadores passaram a dar grande atenção aos jogos de estratégia. Além disso, a semelhança destes jogos com o mundo real possibilita a solução de problemas complexos em ambiente de

jogo. Com isto houve um aumento do número de pesquisadores interessados em utilizar jogos RTS como plataforma de pesquisa, e, consequentemente, o surgimento de espaço exclusivo para a discussão do tema em conferências [Ontanón et al., 2013].

Considera-se importante discutir trabalhos sobre jogos RTS, uma vez que o gênero possui mecânicas que podem ser melhor reproduzidas em jogos MOBA. Além disso, os gêneros possuem algumas similaridades e o número de publicações tendo MOBA como tema exclusivo é escassa.

A pesquisa em jogos RTS possui várias abordagens para o desenvolvimento de agentes. Dentre estas abordagens é comum encontrar dois principais tipos de agentes: os agentes perfeitos e os agentes humanos. Enquanto os agentes perfeitos tem por objetivo masterizar o jogo, jogando de forma perfeita, os agentes humanos tentam mimetizar o comportamento humano dentro do jogo. Em [Weber et al., 2011], propôs-se o desenvolvimento de um agente chamado EISBot, um agente humano desenvolvido para jogar Starcraft. Naquele trabalho, o autor discutiu as características que tornam seu agente mais parecido com um jogador humano. Além disso, discutiu-se os paradigmas de implementação de agentes, como formas homogêneas e heterogêneas. O paradigma homogêneo propõe que o agente deve ser composto por apenas um módulo integrado e responsável por toda a tomada de decisão. Por outro lado, o paradigma heterogêneo diz que um agente deve possuir partes distintas, sendo que cada módulo é responsável por uma área do processo de tomada de decisão.

Mais tarde, o processo de elaboração de agentes por estes paradigmas é revisitado, sendo o heterogêneo utilizado para o desenvolvimento de uma mecânica básica denominada *kiting*. O *kiting* consiste no ato de maximizar o dano realizado enquanto minimiza o dano recebido, executando para isso um processo de bater e correr de forma intercalada. Esta mecânica foi desenvolvida inicialmente em [Uriarte & Ontañón, 2012] e é uma das mecânicas básicas presentes em jogos MOBA. Além disso, no mesmo trabalho os autores apresentaram o uso de Mapas de Influência, técnica introduzida por Tozour [2001] e que consiste na análise das características do mundo de forma discreta. Esta discussão vem da expansão de um trabalho anterior realizado por Hagelbäck & Johansson [2008b], onde o autor discutiu o uso de técnicas de análise para o posicionamento de agentes em jogos de estratégia. Entretanto, acredita-se que o trabalho de implementação do mecanismo de *kiting* poderia ser facilitado se implementado utilizando um jogo MOBA como plataforma de testes, uma vez que foi necessário o isolamento de unidades RTS para os experimentos feitos no trabalho.

Voltando-se para os métodos bio-inspirados, há o trabalho de Liu et al. [2014], onde o autor propôs um sistema capaz de aprender mecânicas básicas de *micromanagement* utilizando RTS como plataforma de testes. Estes comportamentos foram

desenvolvidos por meio de ajustes de parâmetros utilizando Algoritmos Genéticos (do inglês, *Genetic Algorithms*) (GA), técnica proposta por Holland [1973]. Conforme os parâmetros eram ajustados pelo GA, o algoritmo era capaz de assimilar uma mecânica diferente, ajustando assim a mecânica a ser desenvolvida. Mais recentemente, o trabalho de Tavares et al. [2014] propôs o uso de métodos bio-inspirados para a alocação de tarefas em relação aos agentes em jogos RTS, para tanto, utilizou-se o algoritmo *Swarm-GAP*. O sistema desenvolvido foi capaz de alocar tarefas para múltiplas unidades de jogo seguindo suas especialidades, utilizando o jogo Starcraft como plataforma de testes. No campo de análise de jogabilidade, é possível encontrar o trabalho de Stanescu et al. [2013], onde foram analisados os resultados de combates entre grupos de agentes. Estes combates ocorreram em uma versão reduzida do ambiente do jogo Starcraft. O objetivo do trabalho foi prever o resultado de lutas de grupos de agentes. Posteriormente, o trabalho de Stanescu et al. [2014] analisou o balanceamento de jogos RTS a fim de analisar o *level design* de jogos RTS, tendo maior compreensão do *design* deste gênero de jogo. O trabalho foi ainda expandido, demonstrando que seu modelo poderia ser utilizado em jogos de combate em geral. Um ponto interessante é que as técnicas desenvolvidas em ambos os trabalhos são apontadas como promissoras para o uso em jogos MOBA.

A pesquisa sobre jogos MOBA é um tópico bastante recente, portanto há escassa literatura e pequeno número de pesquisadores trabalhando neste tema [Drachen et al., 2014]. Acredita-se que esta escassez se deva principalmente ao fato de o gênero MOBA ser muito recente, tendo seu início em 2009. E, mais do que isto, a maioria dos MOBAs são jogos comerciais, dificultando o uso dos mesmos como ferramenta de pesquisa.

Analisando os trabalhos que utilizam MOBAs como tema central de pesquisa, nota-se a predominância de trabalhos teóricos, como aqueles que realizam a análise do jogador, análise de e-Sports e trabalhos descritivos. O trabalho de Nosrati & Karimi [2013], assim como o trabalho de Rioult et al. [2014], apresentaram uma introdução ao gênero MOBA, discutindo suas características básicas. Já no trabalho de Ferrari [2013], encontrou-se uma discussão detalhada do gênero MOBA, abordando características do *game design* e do gênero como e-Sport.

No trabalho de Drachen et al. [2014], havia uma análise da habilidade dos jogadores de MOBA, verificando o desempenho destes jogadores durante as partidas. O jogo Dota 2 foi utilizado como plataforma de testes, utilizando um banco de dados de *replays* e focando na análise de jogo e visualização de dados. Ainda no campo de análise de jogadores, é possível encontrar o trabalho de Yang et al. [2014], onde os autores procuram identificar as principais ações e funções dentro de um jogo de MOBA. Os autores transformaram os *logs* em grafos temporais, posteriormente aplicando técnicas

de aprendizado de máquina para classificação e identificação dos principais pontos da partida. Em ambos os trabalhos os autores indicaram que o conhecimento gerado poderia ser utilizado para criar agentes em trabalhos futuros.

Outra importante característica dos MOBAs é a formação dos times. Antes do início de cada partida, cada jogador deve escolher um herói a ser jogado durante toda a partida. Este foi o tema de pesquisa no trabalho de Pobiedina et al. [2013], onde os autores executaram uma pesquisa quantitativa relacionando a formação do time e a taxa de sucesso no jogo Dota 2. Além disso, os autores executaram a análise de cada um dos jogadores em particular, concluindo que a experiência e o comportamento desses jogadores durante a partida foram as características mais importantes para levar um time à vitória.

Mais recentemente, o trabalho de Silva et al. [2015] usou o jogo DotA como plataforma para executar o balanceamento dinâmico de dificuldade. Nesta pesquisa, os autores implementaram um agente inteligente com diferentes níveis de habilidade, que são ativados conforme o desempenho do oponente. Numa extensão desse trabalho, Silva et al. [2016], os autores descreveram o mecanismo de um jogo MOBA em detalhes, discutindo suas características. Finalmente, o trabalho de Willich [2015] teve como tema central o desenvolvimento de um agente para jogos MOBA baseado na técnica de Aprendizado por Reforço. O autor usou um arcabouço de aprendizado por reforço, objetivando o aprendizado do agente num ambiente MOBA, utilizando o jogo Heroes of Newerth como plataforma.

Relativo ao objetivo secundário deste trabalho, é possível encontrar o trabalho de Cunha et al. [2015] onde os autores implementaram um agente que trabalhou para auxiliar novos jogadores no aprendizado das mecânicas básicas de um jogo RTS através de um sistema de dicas.

2.4 Sumário

Apresentou-se neste capítulo o gênero de jogo MOBA, um dos mais jogados no mundo na atualidade. Discorreu-se sobre suas características, popularidade e jogabilidade. Além disso apresentou-se o jogo League of Legends, plataforma de testes selecionada para este trabalho.

Discutiu-se ainda trabalhos que discorreram sobre MOBA e RTS, sempre relacionando-os aos problemas encontrados no gênero MOBA. Com estes, identificou-se que existe uma escassez de trabalhos que utilizam MOBA como tema central, o que torna a pesquisa sobre o gênero interessante e inovadora.

Capítulo 3

Metodologia

Neste capítulo é apresentada a metodologia realizada neste trabalho. Ao longo deste discute-se as técnicas implementadas, como Máquinas de Estados Finitos, Planejamento Baseado em Objetivo e Árvores de Comportamento. Discute-se ainda a técnica de Mapa de Influência, responsável pela análise tática para posicionamento e definição das ações do agente. Além disso, são apresentados os desafios envolvidos na implementação destas técnicas, como a mesclagem de valores: grande responsável pelo sucesso do agente ou pela criação de ótimos locais. Finaliza-se com o problema de seleção da plataforma de pesquisa, apresentando-se as razões pelas quais se selecionou LoL para o desenvolvimento deste trabalho.

3.1 Técnicas

Apresenta-se nesta seção as técnicas utilizadas na implementação dos agentes desenvolvidos nesta dissertação. As subseções aqui presentes seguem a ordem cronológica de implementação das técnicas neste trabalho.

3.1.1 Máquina de Estados Finitos

Uma Máquina de Estados Finitos (do inglês, *Finite State Machine*, ou FSM) é uma técnica popular para a representação de estado de conhecimento e tomada de decisão de um agente. Tal popularidade se deve, principalmente, à facilidade de implementação desta técnica. Uma FSM consiste em uma máquina teórica que pode possuir apenas um estado ativo por vez, tendo à sua disposição um número determinado de estados. Para que se possa realizar a mudança de um estado para outro existem as transições, que

consistem em condições que, quando satisfeitas, ativam um novo estado na máquina [Millington & Funge, 2012].

Apesar de sua abstração e implementação simplificada, a FSM não possui boa escalabilidade, sendo necessários vários ajustes sempre que um novo estado é acrescentado. Devido à esta baixa capacidade de expansão, FSMs são aconselháveis para projetos menores, que possuem poucos estados e poucas transições. Na Figura 3.1, observa-se um diagrama exemplificando uma FSM.

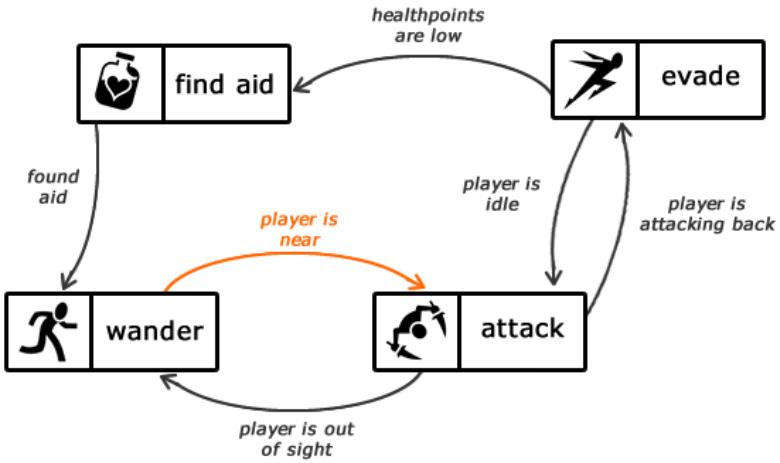

Figura 3.1. Representação de uma FSM de um agente inteligente [Bevilacqua, 2013].

Comumente, uma FSM é modelada como um grafo, onde os nós representam estados e as transições são representadas pelas arestas. As arestas podem representar séries de condições, eventos ou qualquer estrutura de tomada de decisão que possa ser ativada a fim de modificar o estado atual da máquina [Bevilacqua, 2013]. Inicialmente, implementou-se uma FSM em ambos os agentes desenvolvidos nesta pesquisa. Todavia, nota-se que esta técnica é pouco flexível para a realização de análises complexas, devido à grande expansão do número de arestas. Quanto ao agente de suporte ao jogador, percebe-se que seu comportamento é demasiado complexo, e portanto passível da utilização de uma técnica que permite uma rápida prototipação e menor manutenção do código.

3.1.2 Planejamento de Ações Baseado em Objetivo

O Planejamento de Ações Baseado em Objetivo (do inglês, *Goal Oriented Action Planning*), ou simplesmente GOAP, é uma técnica que permite ao agente selecionar quais ações realizar baseando-se em um dado objetivo. O agente analisa o estado do ambiente

e seu próprio estado a fim de eleger possíveis ações a serem realizadas, tendo sempre em vista o objetivo a ser cumprido pelo agente. A seleção das ações a serem realizadas leva em conta o conjunto de ações disponíveis e pré-programadas. O agente, portanto, torna-se capaz de resolver problemas muito mais complexos com esta técnica, já que tem a liberdade de realizar diferentes conjuntos de ações a fim de atingir um mesmo objetivo. Além disso o agente apresenta um comportamento não estático, diferente do encontrado em Máquinas de Estados Finitos.

Esta técnica foi originalmente idealizada para ambientes de jogos digitais, sendo proposta por Orkin [2004]. O autor, por ser também desenvolvedor, implementou esta técnica em alguns jogos comerciais, como No One Lives Forever e F.E.A.R. [Orkin, 2006]. O GOAP consiste em programar uma série de ações que podem ser realizadas, incluindo a estas ações uma série de *pré-condições* e *efeitos*.

Pré-condições são descritores que ditam normas que devem ser satisfeitas para que uma ação seja realizada. Por sua vez, efeitos são mudanças que são executadas no estado do mundo, uma vez que uma ação é executada corretamente. O agente deve utilizar um algoritmo de busca para poder encontrar um conjunto de ações a ser realizadas a fim de se atingir um objetivo. Durante a realização da busca, um grafo dinâmico é gerado, baseando-se nas pré-condições e nos efeitos das ações e no estado do mundo atual.

Nos ambientes em que se encontra a aplicação desta técnica, é muito comum a utilização da busca heurística. Isto se deve à melhor *performance* apresentada quando a busca se beneficia de informações do ambiente para realizar a busca por ações. Em seu pior caso a busca heurística se comportaria como uma busca não-informada, onde teria de expandir todas as possíveis ações, a fim de identificar um conjunto que realizasse o objetivo do agente em questão. A heurística portanto guia o agente em sua busca por ações de forma diferenciada. É possível, por exemplo, observar comportamentos emergentes ou diferentes comportamentos baseados na heurística escolhida. Em [Orkin, 2004], o autor utiliza o algoritmo A* para a realização da busca, demonstrando sua eficácia e discutindo ações emergentes dos agentes.

Para realizar a busca, além da característica da heurística, é importante que se discuta a direção da busca. Enquanto a busca progressiva pode resultar em uma expansão de todos os estados, nota-se que a busca regressiva tem um desempenho muito melhor. Isso se deve ao fato de se partir do pressuposto de que o estado a ser atingido seja realizado por um efeito, e portanto a regressão destes efeitos pode ser inferida muito mais facilmente pelo algoritmo de busca.

Outra importante discussão no contexto do GOAP é a representação do estado do mundo. Em ambientes simples, é perfeitamente aceitável monitorar todas as variáveis

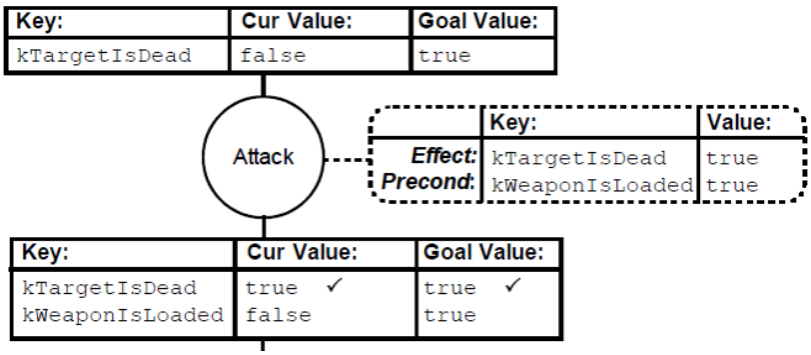

Figura 3.2. Representação de um GOAP como proposto em [Orkin, 2004]. A tabela com linhas sólidas representa o estado do mundo. O círculo representa uma ação executada. Após a execução da ação os efeitos são adicionados ao estado atual do mundo. As pré-condições e efeitos da ação de ataque estão representadas pela tabela de linha tracejada [Orkin, 2004].

do ambiente e realizar modificações nestas por meio de simulações. Todavia, devido à complexidade dos jogos atuais, e também do contexto deste trabalho, é inviável monitorar tantas variáveis. A solução proposta em [Orkin, 2004] é que o ambiente seja representado de forma simplificada e a representação expandida conforme necessário. O ambiente então, assumindo que uma busca regressiva esteja sendo realizada, é representado apenas pelas variáveis do estado objetivo, e o algoritmo de busca deve então tentar inferir quais ações poderiam ter levado àquele estado. Conforme as ações são realizadas, suas variáveis de pré-condição e efeito são dinamicamente alocadas na representação do mundo, e as modificações são aplicadas conforme o comportamento dado de cada uma das ações escolhidas.

A representação então pode ser dada na forma de uma estrutura de dados capaz de armazenar tuplas. Nestas tuplas, o algoritmo realiza as modificações dadas pelas ações e é capaz de armazenar o valor de uma ou mais variáveis, bem como seus identificadores. Uma ilustração desta estrutura de dados pode ser observada na Figura 3.2. Nesta figura, o estado atual é representado por uma tabela de tuplas (linhas contínuas) de valor atual e valor de objetivo. As ações são apresentadas na forma de círculos e atreladas a estas ações há uma tabela de pré-condições e efeitos (linhas tracejadas), que devem ser observados para a execução das ações correspondentes.

3.1.3 Árvore de Comportamento

A Árvore de Comportamento (do inglês, *Behavior Tree*, ou BT) é uma técnica que mimetiza o comportamento de tomada de decisão utilizando uma estrutura de verificação composta em forma de árvore. Este modelo foi desenvolvido devido à necessidade de

uma técnica que tivesse grande poder de escalabilidade para a realização de tomada de decisão em agentes e sistemas. Enquanto numa FSM o agente deve realizar ações dado o seu estado atual, a BT realiza uma série de verificações e retorna um conjunto de ações baseando-se nas verificações realizadas.

Devido à estrutura modular encontrada na BT, sua escalabilidade e reusabilidade de código tornam possível a implementação de comportamentos complexos. Além disso, por meio de estruturas específicas, é possível fazer com que o comportamento programado em uma BT seja determinístico ou não, conforme a inserção de aleatoriedade em suas decisões. Mais ainda, o processo de depuração de comportamentos em uma BT é simples, e os comportamentos emergentes podem ser anulados por meio de tal depuração, tornando a BT uma técnica largamente utilizada para a implementação de inteligência em NPCs na indústria. Finalmente, BTs podem ser modeladas sem a utilização de código, o que permite *game designers* e profissionais não versados em programação a utilizarem e compreenderem a técnica sem muito esforço. O detalhe de uma Árvore de Comportamento pode ser visualizado na Figura 3.3.

Uma BT é uma estrutura modular, e portanto necessita de estruturas de controle que possam conectar suas ações. As principais estruturas encontradas em uma BT são: *Ações*, *Composições* e *Decoradores* [Millington & Funge, 2012].

Ações são os nós da árvore que executam ações ou realizam perguntas. Estes nós normalmente retornam uma resposta, tanto no caso de ações quanto no caso de perguntas. Estas respostas indicam o sucesso de realização de uma ação. Os três tipos de resposta comumente encontrados em um nó de ação são: EXECUTANDO, SUCESSO e FALHA. Ao retornar EXECUTANDO, a BT informa que a ação ainda está sendo executada. A resposta SUCESSO é retornada quando uma ação foi realizada perfeitamente. Por outro lado, ao acontecer uma falha na execução da ação, a BT retorna FALHA. No caso de a ação consistir em uma pergunta, a BT retornará um valor booleano conforme as condições analisadas.

Composições são nós que permitem o aninhamento de outros nós. Existem dois principais tipos de composições: *sequenciadores* e *seletores*. Sequenciadores executam dois ou mais nós em sequência, seguindo sempre a ordem de inserção dos nós. Se algum dos nós retornar FALHA, o nó sequenciador imediatamente retorna FALHA. Caso todos os nós executem perfeitamente, o sequenciador retorna SUCESSO. Os nós seletores executam dois ou mais nós em sequência, porém, diferentemente do sequenciador, quando um dos nós returna FALHA, o seletor continua a execução, tentando executar um próximo nó. Este nó, portanto, só retorna FALHA caso todos os nós filhos retornem FALHA. Em analogia aos sistemas digitais, pode-se dizer que os nós sequenciadores e seletores equivalem às estruturas AND e OR, respectivamente.

Decoradores são nós que auxiliam na modelagem dos sinais retornados ou em repetição de tarefas. Os tipos mais comuns de nós decoradores são: *repetições* e *inversores*. Repetições consistem em repetir uma tarefa um determinado número de vezes ou enquanto uma condição não for satisfeita. Já os inversores são responsáveis por inverter o sinal recebido, semelhante a uma função NOT. É possível ainda encontrar outros decoradores, como geradores de números aleatórios.

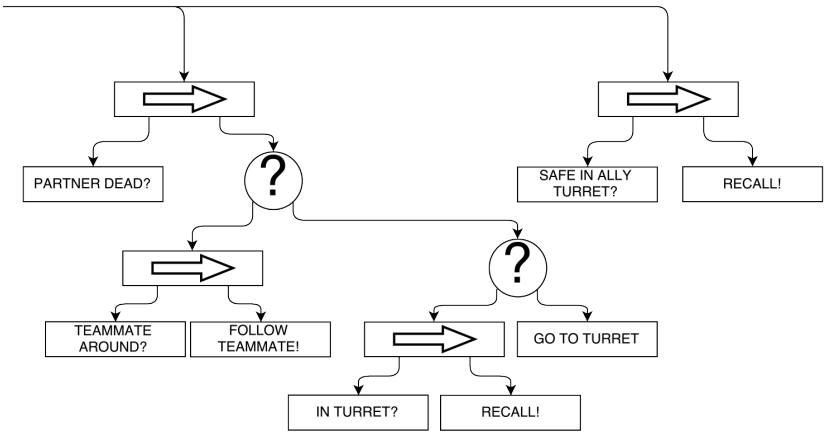

Figura 3.3. Detalhe de uma Árvore de Comportamento desenvolvida para este trabalho. É possível observar sequenciadores (retângulos com seta), seletores (círculos com interrogação) e nós folha de questões (retângulos com comandos terminados em "?") e ações (retângulos com comandos terminados em "!").

A técnica de BT foi a solução final encontrada para a implementação do agente de suporte ao jogador, devido à complexidade do comportamento. A técnica foi implementada utilizando a linguagem Lua, que implementou os comportamentos do agente dentro do jogo League of Legends. Os fatores que levaram à seleção desta técnica foram a escalabilidade, prototipação rápida, modularidade e a facilidade de implementação.

3.2 Análise Tática

Apresenta-se nesta seção as técnicas utilizadas para análise de ambiente de jogo. Primeiramente, apresenta-se a técnica de Mapa de Influência e a técnica de Campos Potenciais. Posteriormente, discute-se os métodos de mesclagem de valores, discutindo o funcionamento destes métodos e seus efeitos sobre os mapas gerados.

3.2.1 Mapas de Influência

Mapa de Influência (do inglês, *Influence Map*, ou IM) é uma técnica que permite a análise do ambiente de forma discreta. Inicialmente esta técnica era largamente aplicada apenas em jogos de Estratégia em Tempo Real, sendo posteriormente aplicada em vários outros gêneros de jogos [Tozour, 2001].

Esta técnica consiste em discretizar o ambiente em partes menores a fim de obter informação sobre cada bloco em específico. Após realizar uma análise sobre cada bloco, executa-se uma atribuição de peso ao bloco analisado. Finalmente, este peso pode ser distribuído utilizando equações matemáticas ou técnicas de dispersão, como métodos Gaussianos. A análise pode consistir de avaliações do terreno, de unidades ou de qualquer outra característica do jogo que possa ser de importância para o conhecimento de agentes. Além disso, um IM pode conter múltiplas camadas, sendo que cada uma destas camadas engloba características de unidades específicas para análise mais detalhada.

A discretização do ambiente de jogo pode ocorrer por meio de diversas técnicas. A mais comumente encontrada é a divisão em *grids*, onde o ambiente deve ser dividido em células de igual tamanho e teor. Outras técnicas que podem ser utilizadas para a divisão do ambiente são: Voronoi, Grafos de Área e *Waypoints* [Champandard et al., 2011]. A primeira técnica no entanto é a mais utilizada devido à sua uniformidade e tamanho dos blocos. Além disto, as técnicas de grafo de área e grafo de *waypoints* estão normalmente atreladas ao *pathfinding*, e associar o processo de tomada de decisão a estes pode gerar um desempenho pior, tanto do ponto de vista analítico quanto do ponto de vista de recursos computacionais [Millington & Funge, 2012]. É necessário ainda verificar o ruído, uma vez que as técnicas de discretização cobrem diferentes áreas. Quanto maior for a área coberta pelo bloco, maior é a chance de que haja ruído ou perda de informação no ambiente.

Após a realização do processo de discretização, o processo de atribuição de pesos é pontual, atribuindo pesos apenas aos blocos onde o elemento a ser analisado se encontra. Todavia, apenas o processo de atribuição pontual não é suficientemente adequado para o processo de análise de características ou de um ambiente. Portanto, como próximo passo, é executada a dispersão dos valores atribuídos ao mapa. Utiliza-se então um método e equação específica para dispersão dos pesos sobre a distribuição discreta do ambiente. Estes métodos comumente são compostos de três parâmetros: *momentum*, *decay* e *frequência de atualização*. Com estes parâmetros, é possível ajustar o tempo de dispersão e o tempo de duração do dado no ambiente. Assim, a técnica de Mapa de Influência se torna versátil, podendo ser utilizada para predição ou análise de

dados históricos [Champandard et al., 2011].

O parâmetro de *momentum* controla a persistência de dados no ambiente. Este parâmetro consiste em uma variável com valores entre 0.0 e 1.0. Quando ajustado em valores altos, o *momentum* faz com que o IM salve dados por mais tempo, permitindo a criação de dados históricos. Por outro lado, valores baixos deste parâmetro fazem com que os dados se percam com mais facilidade, fazendo com que estes sejam menos persistentes. A menor persistência de dados permite à técnica avaliar reativamente o ambiente, uma vez que dados anteriores raramente são encontrados, permitindo menor interferência dos momentos anteriores sobre o cenário atual.

O *decay* controla a o nível de dispersão do peso em relação ao mapa. Um multiplicador é utilizado como parâmetro para a dispersão de valor, aumentando seu alcance. Este parâmetro é comumente associado ao desfoque de valores de peso atribuídos a uma célula, dispersando este peso para as células vizinhas. O *decay* é normalmente encontrado na forma de uma constante, que é subtraída do valor a ser disperso a cada iteração. O ajuste deste parâmetro requer a análise do tamanho do mapa e a distância a ser alcançada pela dispersão do peso em questão. Pequenos valores de *decay* fazem com que a dispersão seja curta, ou seja, o alcance da influência é curto. Já no caso de valores altos, fazem com que o alcance de um peso faça com que a dispersão seja contínua e alcance longas distâncias.

O parâmetro de *frequência de atualização* do IM é fortemente depende da plataforma e do ambiente em questão. É importante que seja feita uma avaliação do *framerate* do ambiente em que a técnica é aplicada. Caso haja uma taxa baixa de quadros, o mapa deverá ser atualizado com uma menor frequência a fim de compactuar com o ambiente. Por outro lado, altas taxas de atualização nem sempre requerem que o mapa seja atualizado. É importante salientar que a atualização de um IM é custosa e pode sobrecarregar o sistema de jogo. Finalmente, este parâmetro pode sofrer com ruídos, como *lag*, *delays*, travamentos, dentre outros, tornando-o mais sensível a ser ajustado.

Existem algumas variações da técnica tradicional de IM, variando o modo de dispersão e inserção de peso sobre um mapa discreto. A técnica de campos potenciais (do inglês, *Potential Fields*) (PF), executa operações similares às realizadas pelo mapa de influência. É importante observar que a técnica de campos potenciais aqui referida não é a mesma utilizada na área de Robótica. Enquanto na Robótica um campo de vetores é gerado, em jogos consiste em inserir e difundir pesos sobre um mapa discreto. A principal diferença entre os campos potenciais e os mapas de influência é o método de dispersão dos valores. Enquanto o mapa de influência insere um valor em uma célula e difunde utilizando equações, o campo potencial normalmente utiliza uma função de

distância Euclidiana para o decaimento dos valores. Além disso, o campo potencial opta por limitar o alcance da difusão, utilizando a mesma técnica de distância. Em Hagelbäck & Johansson [2008a,b], os autores demonstram a eficácia da técnica de campos potenciais em propagar informação sobre uma versão discreta de um mapa de jogo. Para isto, utilizam campos potenciais para inserir informações táticas sobre mapas discretos, utilizando plataformas como ORTS e Starcraft.

Como discutido, campos potenciais possuem um método diferente de distribuição de peso sobre o mapa. Normalmente, o agente dispersa sua influência sobre as células vizinhas, sempre levando em conta o seu próprio alcance. Para tanto, utiliza equações predefinidas para o cálculo do peso da célula, normalmente em função da distância. Devido à característica de coleta de células uma atualização localizada é possível levando em conta os pesos já distribuídos pelos demais agentes. Além disso, não é necessário um ajuste fino de vários parâmetros para a obtenção de uma dispersão controlada, nem o uso de um algoritmo de oclusão para a limitação do alcance da influência. Esta técnica de dispersão foi a escolhida para efetuar a dispersão dos valores sobre o mapa nesse trabalho.

O peso modelado neste trabalho representa o *aggro*, ou seja, a agressividade presente em determinadas células. O *aggro* possui diferentes interpretações conforme o contexto e o jogo em que é utilizado, todavia em todos eles modela o risco de se ocupar um espaço no jogo. No modelo aqui apresentado, pesos positivos representam células mais desejáveis, enquanto pesos negativos representam células perigosas, com alto teor de agressividade. O agente portanto deve evitar comandar o herói controlado por ele para estas células, a fim de minimizar o risco de sofrer quaisquer tipos de agressividade.

3.2.2 Combinação de Valores

Tanto a técnica de Campos Potenciais quanto a técnica de Mapas de Influência têm como cerne a inserção de valores em um mapa discreto que represente o ambiente. Estes mapas podem ser combinados a fim de gerar um mapa de camada única, sendo mais facilmente manipulável, tanto para o agente quanto para o desenvolvedor. Discute-se então as técnicas de mesclagem dos valores inseridos no mapa de influência, a fim de gerar uma única camada.

Após a etapa de coleta de informação, é necessário que o algoritmo realize a operação de combinação de valores, a fim de obter um valor único que possa ser interpretado pelo agente. Em geral, cada unidade mantem um mapa de influência, a fim de realizar suas próprias operações sobre o mapa. A manutenção de múltiplos mapas, porém, é custosa, principalmente em ambientes de tempo real. Em seu trabalho,

Tozour [2001] demonstrou que cada unidade age como se tivesse múltiplos mapas, e o algoritmo deve atualizar todos estes mapas. Além disso, foi discutida a possibilidade da adoção de um mapa único, compartilhado por todas as unidades, sendo esta a técnica adotada no presente trabalho.

O IM apresentado neste trabalho é a forma discreta de um mapa de jogo utilizando a forma de *grid*. Portanto, a representação deste mapa é feita por uma matriz W de valores inteiros de tamanho m x n, onde m é a número de linhas e n o número de colunas da matriz.

A mesclagem de valores em um mapa de influência é responsável por fazer com que a técnica atinja um bom desempenho, diminuindo o número de camadas a ser processadas. O método básico consiste em realizar uma soma das camadas do mapa. Apesar de esta técnica parecer uma solução adequada, ocasionalmente pode gerar uma *performance* irracional ao agente, devido ao grande número de ótimos locais gerados. Dada a representação em forma de matriz apresentada, a forma aditiva desta matriz é dada pelo cálculo das somas das células. Seja w_{ij} uma célula onde i representa a linha e j a coluna. Seja a um peso atribuído por um elemento de um time A sobre uma área limitada células e b o peso atribuído por um elemento B sobre uma área limitada de células. Então, um mapa de influência aditivo pode ser obtido aplicando-se a operação $\forall w_{ij} \in W, w_{ij} = a_{ij} + b_{ij}$. Um exemplo de mapa de influência aditivo pode ser encontrado na Figura 3.4.

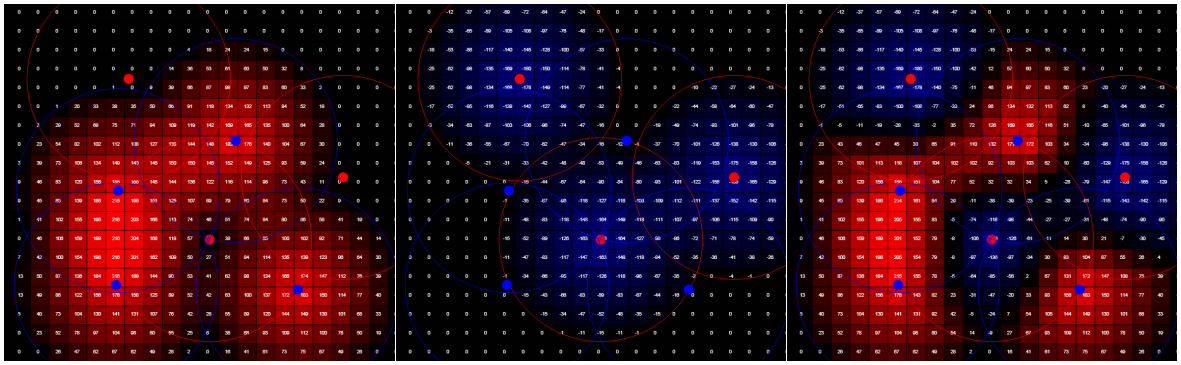

Figura 3.4. Peso dos aliados (esquerda); peso dos inimigos (centro); soma dos aliados e dos inimigos (direita).

Apesar de útil, apenas a operação de soma de valores em um IM não é suficiente para a extração de toda informação necessária. Portanto, executa-se a análise de mais de uma operação. Tais operações se justificam quando se deseja obter mais informações sobre o ambiente ou quando é necessária a mitigação de ótimos locais.

Aos mapas gerados de operações compostas se atribui nomenclatura coerente com

a informação fornecida. O mapa de tensão é aquele que fornece informação sobre a área coberta por todas as unidades do jogo. Para obtê-lo basta executar a operação $\forall w_{ij} \in W, w_{ij} = abs(a_{ij}) + abs(b_{ij})$, obtendo a soma dos valores absolutos das influências. Esta informação é útil durante a análise da área explorada pelos times, além das estratégias utilizadas.

O mapa de vulnerabilidade é aquele que demonstra as fronteiras entre as unidades, demonstrando as áreas onde podem vir a acontecer conflitos. Para a obtenção de um mapa de vulnerabilidade é necessária a subtração de um mapa aditivo do mapa de tensão, representado pela operação $\forall w_{ij} \in W, w_{ij} = abs(a_{ij}) + abs(b_{ij}) - abs(a_{ij} + b_{ij})$. Exemplos dos mapas de tensão e vulnerabilidade podem ser visualizados na Figura 3.5.

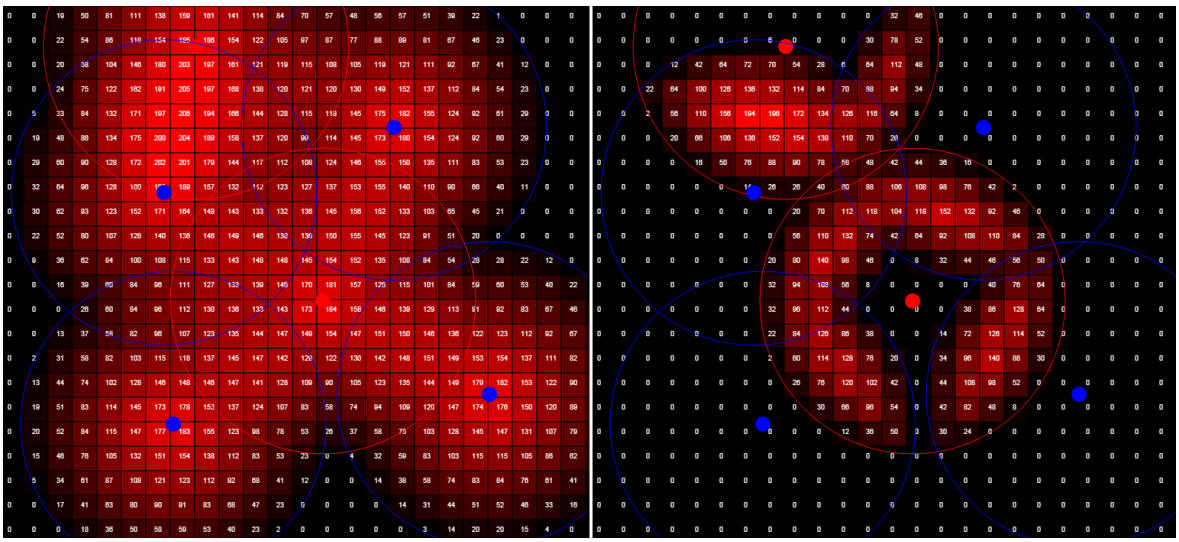

Figura 3.5. Mapa de Tensão (esquerda) e Mapa de Vulnerabilidade (direita)

Como observado, é possível extrair valorosa informação dos mapas calculados quando se utiliza o fator de *aggro*, modelado neste trabalho. É possível, por exemplo, modelar o mapa de influência de modo a predizer locais onde combates possam acontecer. Além disso, modelar o *aggro* resolve uma diversa gama de problemas, como movimentação e combate. Um herói inimigo por exemplo pode emitir um peso positivo ao estar debilitado, podendo ser facilmente abatido. Em situações de cerco, como a apresentada na Figura 3.5, o agente em vermelho poderia abater um inimigo debilitado a fim de facilitar a fuga, facilitando também a tomada de decisão de movimentação. O *aggro*, portanto, se mostra como um fator versátil na modelagem de conhecimento do jogo, uma vez que permite inerentemente a inserção de fatores como risco e recompensa nos pesos atribuídos ao mapa.

3.3 Seleção de plataforma

Selecionar uma plataforma para ser utilizada na pesquisa de Inteligência Artificial pode ser uma tarefa difícil, especialmente no campo de MOBA onde a maioria dos jogos são comerciais. Nesta pesquisa, considerou-se vários jogos durante a fase inicial de seleção da plataforma. Por fim, apenas dois jogos se mostraram suficiente acessíveis para a realização do processo proposto: Heroes of Newerth e League of Legends. Para realizar a escolha entre essas duas plataformas, foi necessário considerar os seguintes requisitos: (a) facilidade de implementação, em termos de linguagem de programação e documentação da API; (b) comunidade, em termos de número de jogadores; e (c) responsividade, como análise do cliente, tempo de resposta, dentre outros.

Heroes of Newerth é um jogo desenvolvido pela *S2 games* e, apesar de ser um jogo comercial, tem o código aberto para desenvolvedores que desejam implementar agentes dentro do jogo. Todavia, há muitas limitações no ambiente de HoN, como a obrigatoriedade do uso de comportamentos pré-programados nos agentes.

League of Legends, desenvolvido pela *Riot Games*, é um jogo comercial e atualmente o jogo mais jogado do mundo, consumindo cerca de 23% da jogabilidade *online* [Murphy, 2015]. A desenvolvedora não disponibiliza o código de League of Legends, portanto não há uma plataforma de programação oficial para o jogo. Todavia, há ferramentas que permitem a injeção de código dentro de LoL como a *LeagueSharp* e a *Bot of Legends*, sendo a última escolhida para o desenvolvimento desta pesquisa. Essas ferramentas são análogas ao comportamento da *BWAPI* no ambiente de Starcraft, fornecendo uma plataforma de desenvolvimento de agentes utilizando a chamada de código do jogo.

Ambas as plataformas utilizam a linguagem de programação Lua [Ierusalimschy et al., 1996], uma linguagem de programação padrão para o desenvolvimento de agentes na indústria [Millington & Funge, 2012]. Apesar disso, a plataforma de desenvolvimento de Heroes of Newerth não possui documentação concreta, sendo composta de diversos *posts* em fóruns de usuários que desenvolvem agentes nessa plataforma. Nas plataformas disponibilizadas para League of Legends, em especial a *Bot of Legends*, há documentação completa, além de enciclopédias e fórums, tornando mais fácil o acesso à informação.

Em relação ao número de jogadores, LoL é mais popular que HoN, principalmente na América do Sul. O tempo médio para encontrar uma partida contra agentes em HoN foi de 15 ou mais minutos. Já em LoL, o tempo aproximado é de 5 segundos. Além disso, HoN apresenta uma alta latência durante as partidas, fato que raramente acontece em League of Legends.

Dadas as condições discutidas nesta seção, selecionou-se o jogo League of Legends juntamente à plataforma *Bot of Legends* para esta pesquisa. É importante salientar que nenhuma das ferramentas oferece vantagens ao agente, tendo exatamente as mesmas informações que o jogador.

3.4 Sumário

Este capítulo teve por objetivo apresentar e discutir a metodologia desenvolvida durante o desenvolvimento deste trabalho. Discorreu-se pelas técnicas, apresentando suas características e implementações dentro deste trabalho. Além disso discutiu-se os problemas envolvidos, como a mesclagem de valores no mapa de influência. Finalizou-se com a discussão de seleção de plataforma, tendo sido selecionado o jogo League of Legends como plataforma de pesquisa para este trabalho.

Capítulo 4

Agentes Inteligentes para Jogos MOBA

Neste capítulo apresenta-se os agentes desenvolvidos utilizando as técnicas apresentadas no capítulo anterior. O primeiro agente inteligente tem por objetivo ser capaz de jogar MOBA, interagindo com o ambiente de jogo e suas características, obtendo o melhor desempenho possível. Posteriormente, apresenta-se um agente capaz de auxiliar novos jogadores durante os períodos iniciais de aprendizado em jogos MOBA. Em ambos os casos, o jogo League of Legends é utilizando como plataforma de testes para os agentes.

Discute-se ainda neste capítulo a arquitetura utilizada em cada um dos agentes implementados. O primeiro utiliza uma arquitetura multi-camadas capaz de fazer com que seus módulos interajam e formem um agente inteligente. Já o segundo agente, que fornece suporte aos jogadores, utiliza uma estrutura acoplada baseada em uma Árvore de Comportamento, oferecendo uma maior consistência comportamental em relação ao primeiro agente.

4.1 Um Agente de Jogo para MOBA

O principal objetivo deste trabalho foi desenvolver um agente capaz de jogar MOBA efetivamente. Isso significa que o agente deve agir de forma racional ao interagir com o ambiente do jogo por meio do controle de um herói do jogo. Além disso, este agente deve ser capaz de interagir com outros agentes de forma racional e, preferencialmente, competitiva. A plataforma escolhida para o desenvolvimento deste agente foi o jogo League of Legends, onde foi implementado um agente baseado numa arquitetura multi-camadas, que será discutido nas próximas seções.

Inicialmente, o objetivo deste trabalho consistia em resolver um problema de movimentação de agentes dentro do contexto de ambientes de combate competitivo. Os jogos MOBA se mostraram então uma plataforma que favorecia este tipo de pesquisa. Todavia o problema de movimentação está fortemente atrelado ao problema de combate, onde um agente pode, por exemplo, abater uma unidade fraca a fim de facilitar a movimentação sobre o terreno de jogo. Por meio da implementação de módulos não acoplados, pôde-se atacar o problema de movimentação e o de combate de forma unitária, gerando comportamentos interessantes ao agente, como o de *kiting*, sendo estes discutidos a seguir.

4.1.1 Agente de Arquitetura Heterogênea

Discute-se nesta seção a metodologia de desenvolvimento do agente capaz de jogar MOBA. Optou-se por utilizar uma arquitetura heterogênea, composta de camadas distintas capazes de filtrar os sinais de entrada e realizar a análise dos mesmos, a fim de se retornar uma ação. Esta arquitetura tem por objetivo coordenar o agente inteligente de forma que o mesmo aja racionalmente e seja capaz de vencer partidas de um jogo MOBA.

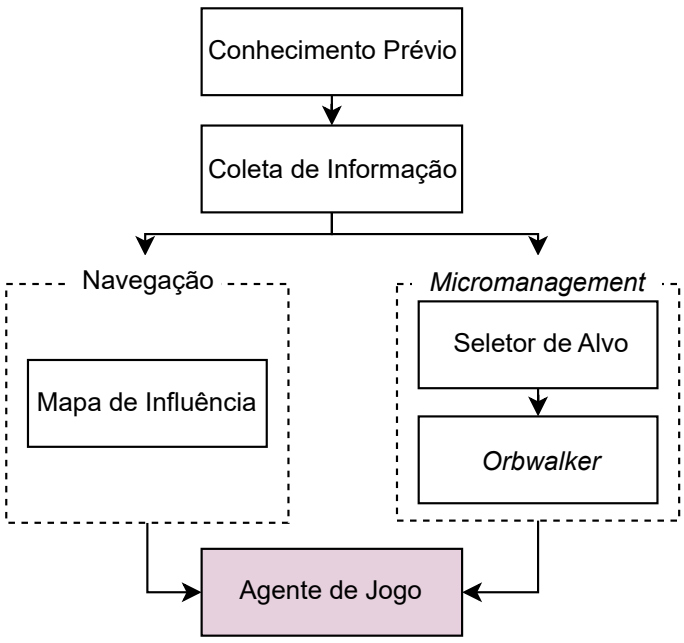

Figura 4.1. Estrutura da arquitetura proposta. Os retângulos com linhas sólidas representam módulos implementados. Os retângulos em linhas tracejadas representam camadas. A análise que é gerada é dada como saída para um interpretador que é responsável por comandar o herói em LoL.

A arquitetura implementa duas camadas distintas e independentes: a camada de *navegação*; e a camada de *micromanagement*. A primeira é responsável por posicionar o agente no ambiente, enquanto a segunda é responsável pelas mecânicas a serem executadas. Uma visão geral desta arquitetura pode ser observada na Figura 4.1. Estas camadas serão discutidas em mais detalhes a seguir.

4.1.1.1 Camada de Navegação

A camada de navegação é responsável pelo posicionamento do herói dentro do ambiente do jogo. Esta camada requer que o ambiente seja analisado de forma minuciosa, procurando identificar áreas de perigo e áreas seguras. Com estas informações o agente pode tomar decisões que sejam favoráveis às decisões táticas, se posicionando de forma a obter a máxima recompensa possível. Em geral, o mapa de influência utilizado é composto por pesos. O agente procura sempre se posicionar no melhor peso possível, considerando que depois de todas as análises, esta é a posição mais favorável ao agente.

A camada de navegação deve receber o mapa de influência completo do jogo a fim de realizar as análises necessários sobre o mesmo, onde analisa os pesos já atribuídos anteriormente. Após o processamento, esta camada retorna uma tupla P na forma $P = \{x, y, z\}$, onde x, y e z são valores reais representando a coordenada tridimensional na qual o agente deve se posicionar. Por se tratar de um sistema em tempo real, foi necessária a execução de testes preliminares com esta camada, a fim de identificar qual seria a frequência de atualização da mesma. Devido ao tamanho do mapa encontrado em LoL, foi fixada uma taxa de três atualizações por segundo do mapa de influência implementado. Esta taxa foi fixada levando em conta que a frequência de atualização não deve causar alta latência ao jogo, consumindo muitos recursos computacionais.

Para computar a tupla a ser retornada, esta camada percorre todo o IM, aplicando os pesos dos agentes segundo as equações que foram atribuídas a cada um destes agentes. Posteriormente, é feita a mesclagem dos valores e finalmente é realizada uma busca na matriz pela célula com o maior peso. As coordenadas desta célula então são retornadas ao agente, que deve se movimentar para a mesma. Não foi necessária a implementação de um sistema de busca de caminho, utilizando-se o sistema contido na plataforma.

4.1.2 Camada de Micromanagement

A camada de *micromanagement* é responsável por identificar quais as ações a serem realizadas pelo agente. Tal raciocínio é necessário porque o agente só pode realizar uma ação de cada vez. Esta camada então pode ordenar o agente a realizar a ação de

mover ou atacar. Além de ordenar estas ações, esta camada é responsável por decidir quais serão os alvos a serem atacados. Portanto, há duas mecânicas encapsuladas nesta camada: o *orbwalking*, responsável pela tomada de decisão entre atacar ou mover; e o *seletor de alvos*, responsável por decidir qual alvo será o foco dos ataques. Discute-se estas mecânicas a seguir.

Orbwalking é um processo criado originalmente em jogos de estratégia em tempo real que consiste em controlar perfeitamente o ataque e movimentação de um herói. Basicamente, na maioria dos jogos de estratégia, os ataques são controlados por animações. Estas animações em certo ponto criam um projétil, e, a partir deste momento, cancelar a animação não prevenirá a criação do projétil de se completar. Os jogadores então exploram esta mecânica a fim de maximizar o número de ataques por segundo, além de poder se movimentar com maior frequência. Para a execução desta mecânica, os jogadores devem executar outra ação que seja capaz de cancelar a animação em execução no personagem. Assim, conseguem aproveitar melhor as mecânicas do mesmo sem perder o tempo de execução do restante da animação.

É importante observar que o conceito de *orbwalking* difere do conceito de *kiting*. Enquanto o segundo tem por objetivo minimizar o dano recebido por meio da execução da mecânica de bater e correr, o primeiro apenas maximiza o desempenho do herói. O *orbwalking* então é simplesmente uma mecânica que deve informar ao agente quando atacar e quando mover, sem se preocupar para onde correr, ou como correr, ou o que atacar.

Neste trabalho, optou-se por gerar uma estrutura de *orbwalking* baseada em análise tática. Portanto, foi necessária a inclusão de informações de análise do agente, além das informações sobre animações. Ações que possam afetar o agente negativamente passam então a fazer parte do processo de tomada de decisão do *orbwalker*, gerando assim uma estrutura que leva em conta a recompensa em atacar ou mover. Como entrada, este mecanismo lê as informações contidas no herói, como velocidade de ataque, execução de ações e tempos de recarga, além de receber informações sobre o mapa de influência, como unidades perigosas. Após processar estas informações, a estrutura retorna uma tupla denominada A na forma $A = \{atacar, mover\}$, onde *atacar* e *mover* são valores booleanos.

O **Seletor de Alvo** é uma estrutura de decisão que consiste em selecionar um alvo a ser considerado. Esta seleção é um dos fatores mais importantes numa situação de combate, fazendo com que o máximo de inimigos possam ser derrotados. Executar esta seleção é um problema em aberto em todos os jogos de combate de múltiplas unidades em tempo real, uma vez que o cenário é muito dinâmico e envolve predições e probabilidades. O trabalho de Uriarte & Ontañón [2012] demonstrou que uma boa

forma de executar a seleção de alvo é se basear em informações táticas. Os autores modelaram um fator denominado *aggro*, que denota a agressividade de um agente inimigo.

Neste trabalho, coleta-se dados referentes ao ambiente, modelando estes dados conforme as equações previamente discutidas. Após este processo, realiza-se a dispersão de pesos no mapa de influência. Finalmente, após esta dispersão, é possível utilizar os dados a fim de selecionar o alvo. Em geral, o agente tem preferência em selecionar heróis, e posteriormente tropas. Necessariamente, o combate a um inimigo sobrepõe o comportamento de *farming*. No entanto, nem sempre o comportamento de combate é assumido, levando em conta o cenário e a fase de jogo. Além do combate, o seletor de alvo é responsável por deliberar quando é importante o foco de torres, executando a ação de controle de objetivos.

Outra ação importante em relação à seleção de alvos é a execução do *farming*. Selecionar qual tropa será o foco do ataque é uma tarefa difícil, mas é essencial para a coleta eficiente de recursos. Além de selecionar a tropa correta, é necessário se certificar que esta tropa esteja ao alcance do agente, tornando o *farming* uma tarefa conjunta das camadas de seleção de alvo e navegação. Todavia, este comportamento não é diretamente implementado, sendo que cada camada assume um papel em específico. A camada de navegação tem preferência automática por se aproximar de tropas com menor HP, enquanto o seletor de alvo procura focar nas tropas com maior probabilidade de morte. A junção destes dois comportamentos faz com que o agente execute o *farming* de forma eficiente.

Outra observação interessante é a emergência de um comportamento de *kiting* tanto em relação às tropas quanto aos campeões inimigos. Enquanto em [Uriarte & Ontañón, 2012] foi necessária a implementação de uma estrutura dedicada para a execução desta mecânica, no presente trabalho o comportamento emerge da própria arquitetura. Os múltiplos mecanismos de tomada de decisão executam uma concatenação das respostas e fazem com que o agente ataque e corra naturalmente, sem precisar ser comandado diretamente para executar esta mecânica.

4.1.3 O Modelo de Conhecimento para Tomada de Decisão Tática

Os métodos utilizados neste trabalho para realizar a análise do ambiente são baseados em análise tática para tomada de decisões. Para tanto foi necessário o desenvolvimento de uma base de conhecimento, obtida através da coleta de dados do ambiente de jogo e de dados disponíveis na página *web* da desenvolvedora de *League of Legends*. Após

a etapa de coleta de dados, estes foram utilizados para gerar equações, as quais foram utilizadas para dispersar os pesos sobre o mapa de influência. Durante a etapa de criação do mapa de influência, o agente leva em conta quatro características básicas: terreno, tropas, torres e heróis. Essas características são classificadas conforme seu nível de dinamismo dentro do *gameplay*. O terreno é classificado como uma característica estática, sendo que esta classificação se deve ao fato de o terreno não poder ser modificado durante o jogo. Tropas e heróis são classificados como características dinâmicas, devido ao fato de estas unidades poderem se mover, atacar e utilizar poderes. Já as torres, são classificadas como características semi-dinâmicas, porque elas não podem se mover, mas possuem uma máquina de estados específica, podendo atacar jogadores e tropas. Cada classe de características tem a sua própria base de conhecimento modelada para a dispersão de pesos específicos sobre o mapa de influência. As informações utilizadas para análise de cada uma dessas classes serão discutidas nas próximas subseções.

4.1.4 Colisores

Para analisar o espaço de jogo, foi implementada uma representação do mapa de forma discreta utilizando um mapa de influência. Devido ao relevo do terreno não influenciar no sistema de combate, não foi necessário uma modelagem avançada com múltiplas camadas. Por isso, a solução encontrada foi discretizar o mapa na forma de um *grid*. Cada célula é representada por um quadrado de mesmo tamanho e teor, e em sua completude, o mapa de influência sobrepõe o mapa de jogo.

O primeiro desafio na criação do mapa de influência se encontra no processo de discretização. O desenvolvedor deve ajustar o parâmetro de tamanho da célula ajustando assim a resolução da mesma em relação ao mapa. Há portanto a necessidade de balanceamento entre uso de recursos computacionais e o nível de informação coletada. Um mapa com células pequenas é capaz de coletar muito mais informação sobre o ambiente, todavia, demanda muito mais recursos computacionais para a sua manutenção. Por outro lado um mapa com células grandes pode gerar ruído devido à sua baixa resolução, porém, utiliza pouquíssimos recursos computacionais. Após testes iniciais, foi fixada uma resolução de 150 pixeis como lado da célula quadrada do mapa de influência. Esta resolução permitiu ao mapa coletar informação suficiente para o processo de tomada de decisão do agente, enquanto usava os recursos computacionais sem causar atrasos no jogo.

O processo de modelagem do terreno dentro do mapa de influência consiste na remoção das células que se localizam dentro de áreas de colisão permanente, como pare-

des e estruturas. A remoção dessas células permite ao agente melhorar sua *performance* geral, reduzindo o tamanho da matriz a ser atualizada. A discretização do ambiente e a remoção das células é executada apenas uma vez no início da partida. Essa decisão foi tomada tendo em vista que MOBAs são jogos dinâmicos, podendo ter seu mapa modificado após uma atualização. É importante observar que em League of Legends acontecem atualizações de 15 em 15 dias, em média, e neste tempo há a possibilidade de mudança no mapa, apesar de essas mudanças não serem comuns. Por isso, optou-se por não armazenar o mapa de forma estática, permitindo assim o melhor ajuste do algoritmo. Além disso, um dos objetivos deste trabalho foi propor algoritmos que possam ser utilizados em diferentes jogos do gênero MOBA. Assim, ao gerar o mapa em diferentes jogos, o mapa de influência automaticamente se adapta ao mapa do jogo. Na figura 4.2 é possível observar uma demonstração do mapa de influência gerado. Nesta figura é possível observar que o mapa de influência gerado exclui as células das áreas onde há colisores do jogo.

Figura 4.2. Mapa de Influência gerado no mapa Summoner's Rift. Observe que as paredes e estruturas não contém células.

4.1.5 Torres

As torres são fortes estruturas defensivas, e são classificadas como características semi-dinâmicas. Essa classificação se deve ao fato de as torres não poderem se mover mas poderem mudar o seu estado, atacando tropas e heróis. Divide-se a análise das torres em dois momentos: primeiramente analisa-se o comportamento das torres inimigas e, posteriormente, o comportamento das torres aliadas.

Uma torre qualquer pode assumir dois estados diferentes: estado agressivo ou estado passivo. A torre assume o estado passivo quando não existe nenhuma unidade do time adversário ao seu alcance. Neste estado, a função da torre é procurar por unidades do time adversário que estejam em seu alcance de ataque. Assim que alguma unidade do time adversário entra em seu alcance, a torre automaticamente assume o estado agressivo.

Durante o estado agressivo, a torre ataca as unidades do time adversário que estiverem em seu alcance. Para tanto, a torre possui um seletor de alvo específico, atacando primeiramente heróis adversários que estejam atacando heróis aliados e posteriormente tropas adversárias em seu alcance. Caso o alvo da torre seja uma tropa e um herói adversário atacar um herói aliado dentro de seu alcance, a torre automaticamente muda o seu comportamento para atacar o herói adversário que demonstrou comportamento agressivo para com o herói aliado, auxiliando-o em sua sobrevivência.

Intuitivamente, a torre em estado agressivo deveria emitir uma influência negativa sobre o mapa, todavia tal intuição nem sempre é correta. Suponha uma situação onde a torre se encontra no estado agressivo, porém o seu alvo é uma tropa. Esta torre não oferece perigo ao herói controlado pelo agente, uma vez que a torre continua a atacar a tropa desde que o agente não ataque um herói inimigo. Neste momento o objetivo do agente pode ser o de focar a torre, causando o máximo de dano possível, enquanto a torre ataca as tropas aliadas ao herói.

A modelagem de influência das torres inimigas é realizada em função do ataque ao agente. Quando há risco de ataque da torre inimiga ao agente, esta torre emite uma influência negativa. De forma análoga, quando não há risco de dano ao herói, esta torre emite uma influência positiva, fazendo com que o agente seja atraído a atacá-la.

É necessária então uma análise do quão seguro é se manter no alcance da torre em função das tropas que se encontram ao alcance da mesma. Inicialmente, a abordagem utilizada baseava-se na soma do HP das unidades ao alcance da Torre. Essa abordagem se mostrou ineficiente, uma vez que o agente continuava a receber dano da Torre. Decidiu-se então pela abordagem de contagem do número de tropas ao alcance da Torre, sendo denominado α. Experimentos demonstraram que quando o valor de α é maior ou igual a 3, o dano recebido pelo agente é reduzido a zero em todos os casos. O valor de α é então utilizado como condição da mudança do estado da torre, modificando assim a influência emitida pela mesma sobre o mapa. Quando o número de tropas é maior do que α, uma zona positiva é gerada em torno da torre, fazendo com que o agente seja atraído para esta zona. Do contrário, uma zona negativa é gerada em volta da torre, fazendo com que o agente seja repelido por esta área.

Além desta análise, é necessário considerar que podem acontecer situações onde o

agente seja atraído em direção à base inimiga. Para resolver este problema são levados em conta os dois fatores: (a) células que se encontram mais próximas da base aliada recebem um peso maior; e (b) células onde o agente pode atacar de seu alcance máximo recebem um peso maior. O fator a faz com que o agente se comporte defensivamente, evitando o comportamento denominado *tower dive*[1]. O item a também beneficia o posicionamento do agente, minimizando suas chances de ser atacado pelas costas. O item b faz com que o agente tire proveito de suas mecânicas, fazendo com que os ataques sejam, preferencialmente, executados do máximo alcance, melhorando assim o seu posicionamento.

Tendo em vista o posicionamento defensivo mais próximo à base aliada e mantendo-se no máximo alcance do qual o agente possa atingir a torre inimiga, foi necessária a criação de um fator que favorecesse esta área. Para tanto, levou-se em conta a distância entre a torre analisada e a base aliada ao agente, representada por d_{tb}, e a distância entre o ponto analisado e a base aliada, representada por d_{pb}. Realizou-se a divisão do valor de d_{tb} pelo valor de d_{pb} a fim de se obter um fator, o qual se denominou τ, que favorece os pontos que estejam mais próximos à base aliada. O cálculo do valor de τ é apresentado na Equação 4.1.

$$\tau = \frac{d_{tb}}{d_{pb}} \tag{4.1}$$

Devido ao tamanho das células, acontecem situações onde o agente pode ficar preso entre duas células e não executar ações. Para mitigar este erro, foi inserida uma variável que flexibiliza o espaço de localização do agente. Detectou-se empiricamente que um valor equivalente à metade da resolução das células é suficiente para resolver o problema de movimentação sem travamentos. Esta variável flexibilizadora foi denominada Δ.

Finalmente, o agente deve ser permitido a tomar decisões arriscadas em relação às Torres. Um exemplo dessas decisões é poder executar ação de *tower dive*, caso tenha certeza de que pode abater um inimigo fraco que esteja no alcance de uma torre inimiga. Para isso, foi implementado um limite da influência negativa difundida pela torre. O limite cria um abrandamento da influência, sendo controlado pela variável ε.

O resultante da junção destas variáveis é uma influência que decai ao longo do alcance de ataque da torre, denominado T_r. São necessárias então duas equações para definir a influência da torre, sendo cada uma das equações responsáveis pelo estado atual da torre. No caso da torre em estado passivo, onde o agente é atraído a atacar

[1]O ato de entrar no alcance de uma torre inimiga e executar ações hostis a um herói inimigo, fazendo com que a torre tenha o atacante como alvo.

a torre, a influência é ainda modificada pelo alcance de auto ataque do herói, representado por H_r. Uma visualização desta influência pode ser observada na Figura 4.3. A Equação 4.2 demonstra a projeção de influência pela torre no estado agressivo com tropas, onde o herói deve se manter no máximo alcance e focar a torre para maior segurança e posicionamento.

$$w_p = \begin{cases} d_{pt}, & d_{pt} < H_r - \Delta \\ \tau * d_{pt}, & H_r - \Delta \leq d_{pt} \leq H_r \\ T_r - d_{pt}, & \text{senão} \end{cases} . \tag{4.2}$$

No caso de a torre estar em modo passivo, não é desejável que o agente comande o herói a entrar no alcance da torre. Para tanto uma forte influência negativa é desprendida pela torre, estimulando o agente a manter distância da área coberta pelo alcance da torre. Todavia é necessária a flexibilidade para jogadas arriscadas, portanto é implementada uma borda, dada por ε. A Equação 4.3 mostra a dispersão de valores pela torre no estado passivo.

$$w_p = \begin{cases} -T_r, & d_{pt} > \varepsilon \\ -\infty, & \text{senão} \end{cases} \tag{4.3}$$

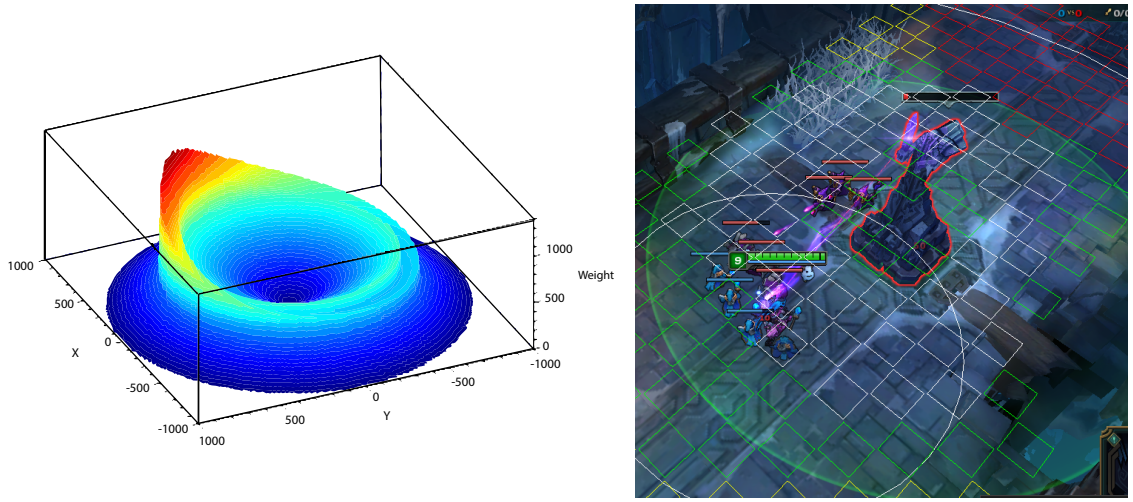

Figura 4.3. A esquerda, representação gráfica da Equação 4.2. Áreas vermelhas são mais desejáveis; áreas azuis são menos desejáveis; a direita, situação do jogo capaz de gerar a equação, o herói é atraido pela torre já que o foco desta torre serão as tropas aliadas

4.1.5.1 Torres aliadas

As torres aliadas são um caso especial de modelagem de torres dentro da arquitetura proposta. Assim como todas as torres do jogo, as torres aliadas podem mudar seu estado conforme as tropas inimigas. Todavia, por serem torres aliadas ao herói controlado pelo agente, não é necessária a implementação diferenciada de estados, uma vez que estas torres não podem causar dano ao herói. A torre aliada provê uma grande área segura para o agente e seus aliados, sendo necessária a sua análise para um melhor posicionamento do agente no ambiente de jogo. Em relação ao posicionamento, é importante salientar que posições avançadas, mesmo dentro do alcance da torre aliada, não caracterizam um cenário seguro para o agente. Por isto, durante a análise, o agente deve considerar tanto o alcance da torre quanto alcance das unidades inimigas, localizando um posicionamento no qual não receberá dano dos inimigos.

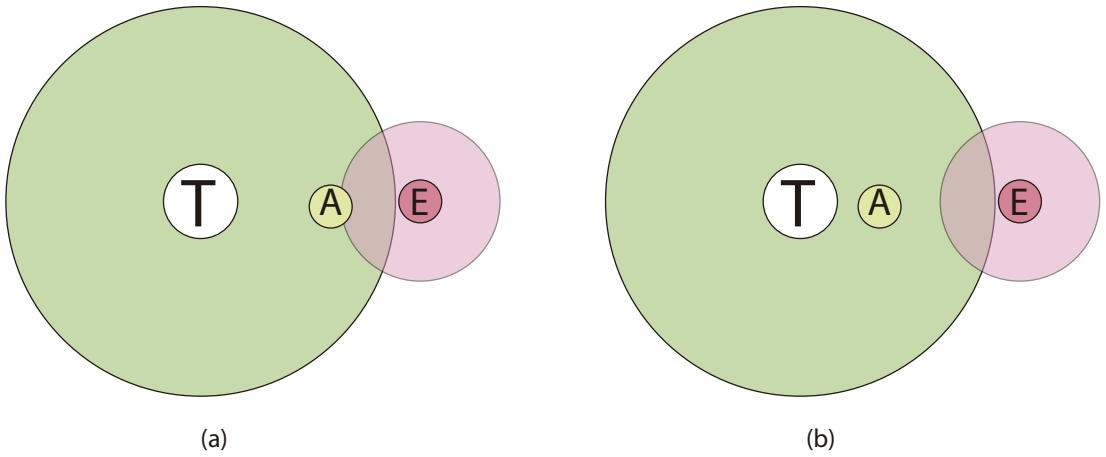

Figura 4.4. A área verde representa o alcance do ataque da torre enquanto a área vermelha representa o alcance do ataque do agente inimigo. Em (a) o posicionamento é desvantajoso para o agente; em (b) o posicionamento é seguro. T representa a torre aliada, A representa o herói controlado pelo agente e E representa um herói inimigo.

No diagrama da Figura 4.4, é possível observar a diferença de posicionamento no ambiente. Na situação (a), o agente pode ser atacado por unidade inimiga mesmo sob cobertura da torre aliada. Por outro lado, na situação (b) o agente não pode ser alcançado, a não ser que a unidade inimiga execute o *tower dive*. Portanto, a segunda situação é mais segura em relação ao posicionamento do agente, concedendo proteção extra devido ao alcance da Torre e ao seu comportamento.

Além da análise de posicionamento, é necessário manter distância da torre, tendo

em vista não colidir com a mesma. A colisão com uma torre é um exemplo de mal posicionamento, uma vez que esta ação pode prejudicar na busca de caminhos ou na movimentação do agente. Além disso, em muitos MOBAs há heróis que possuem habilidades que são ativadas ao executar a colisão de uma unidade com o terreno. Já que a torre é uma estrutura fixa, esta pode ativar as habilidades citadas, sendo um exemplo de mau posicionamento estar alinhado com a torre em relação aos heróis inimigos. Para evitar tal situação, a equação que gera a influência da torre aliada considera o espaço de colisão maior do que a caixa de colisão da torre. É criado então um platô repulsivo em volta da torre aliada. Para tanto leva-se em conta uma distância δ, que determina a distância que o herói deve manter da torre, a fim de evitar colisões. Efetua-se então a atribuição da influência positiva da torre aliada, verificando-se o valor máximo entre a influência atual e a diferença entre o alcance da torre, T_r, e a distância entre o ponto analisado até a torre, d_{pt}. A equação que gera a influência para a torre aliada leva em conta todas as informações apresentadas anteriormente e é apresentada na Equação 4.4.

$$w_p = \begin{cases} max(T_r - d_{pt}, w_p), & d_{pt} > \delta \\ 0, & \text{senão} \end{cases} \tag{4.4}$$

4.1.6 Tropas

As tropas são unidades controladas pela Inteligência Artificial do jogo e são parte importante do *gameplay* de todos os MOBAs. Essas unidades são responsáveis por forçar as rotas em direção à base inimiga, lutando com as tropas inimigas e destruindo estruturas defensivas, quando possível. Além disso, derrotar essas tropas significa obter dinheiro e experiência. Por isto, uma análise efetiva dessas unidades é essencial para o desenvolvimento de agentes inteligentes no ambiente MOBA.

O jogador não tem controle direto sobre as tropas, por isso estas podem ser classificadas como NPCs (do inglês, *Non Playable Character*, ou Personagem não Jogável). As tropas surgem em cada uma das bases em espaços de tempo definidos. Os grupos de tropas que surgem nas bases são comumente chamados de ondas (do inglês, *waves*). Após o surgimento, cada tropa segue um caminho por uma rota específica, lutando com os inimigos ou estruturas que encontrarem pelo caminho, incluindo tropas inimigas, heróis inimigos, torres, e, no caso de League of Legends, inibidores e Nexus. Lembrando que o Nexus é a principal estrutura, e quando capturada concede a vitória ao time que a capturou.

As tropas inimigas geram uma área atrativa de influência, a fim de fazer com que o agente tenha preferência por atacá-las. A dispersão da influência dessas tropas

é proporcional ao alcance do ataque básico do herói selecionado, fazendo com que o agente possa executar a mecânica de *farming* usando seu máximo alcance de ataque. Esta técnica é similar à aplicada por Hagelbäck & Johansson [2008a], mantendo o herói o mais longe possível das unidades inimigas, mas ainda assim mantendo o inimigo ao alcance do ataque.

Como observado, o principal foco de análise em relação às tropas é a execução da mecânica de *farming* para a coleta de dinheiro e experiência. Para tanto, é necessário que o herói execute o último golpe em uma tropa. Na primeira abordagem, foi implementada uma equação que fosse capaz de posicionar o agente em relação ao seu máximo alcance do ataque básico, H_r. Para isso, executa-se uma análise levando em conta a distância do ponto analisado até a tropa em questão, representado por *dpc*. Após gerar a influência até o alcance do herói, é necessário levar em conta o posicionamento defensivo em relação à base aliada, representado por τ. O resultado é uma influência que favorece o agente à comandar o herói a executar o *farming* de forma defensiva. A Equação 4.5 demonstra a geração de influência pelas tropas.

$$w_p = \begin{cases} d_{pm}, & d_{pc} < H_r - \Delta \\ max(\tau * d_{pc}, w_p), & \text{senão} \end{cases} \tag{4.5}$$

Observou-se que essa abordagem não executava o *farming* de forma adequada. Concluiu-se então que o modelo necessitava de algum estímulo para selecionar as tropas com menor HP. O jogo fornece informação sobre HP atual das tropas, assim, implementou-se análise da porcentagem de HP em cada uma das tropas. Quanto menos HP, maior é a influência atribuída por essa tropa ao mapa de influência. Essa relação é armazenada na variável ϕ inserida na equação anteriormente modelada. A equação resultante (Equação 4.6) demonstrou maior eficiência na *performance* do *farming*.

$$w_p = \begin{cases} d_{pm}, & d_{pm} < H_r - \Delta \\ max(\tau * (d_{pc} + 100 - \phi), w_p), & \text{senão} \end{cases} \tag{4.6}$$

Um desafio enfrentado durante a implementação da influência das tropas é o método de mesclagem do valor das influências. Cada uma das tropas atribui um valor de influência a uma célula e, posteriormente, essa influência é difundida pelo mapa. Ocasionalmente, ocorre a sobreposição de áreas cobertas por estas tropas, já que há uma grande quantidade de tropas no jogo. O método inicial consistiu em somar as influências destas tropas. Verificou-se então que o método da soma é uma abordagem ineficaz, já

que há a geração de múltiplos ótimos locais[2] no mapa de influência. Verificou-se então que a melhor abordagem a ser utilizada é a seleção do maior valor encontrado na célula, já que os valores menores são menos interessantes para o agente. Além disso, a utilização do máximo valor de uma célula preserva os valores úteis atribuídos ao mapa, evitando a geração de ótimos locais no processo de mesclagem. Uma comparação dos métodos pode ser observada na Figura 4.5.

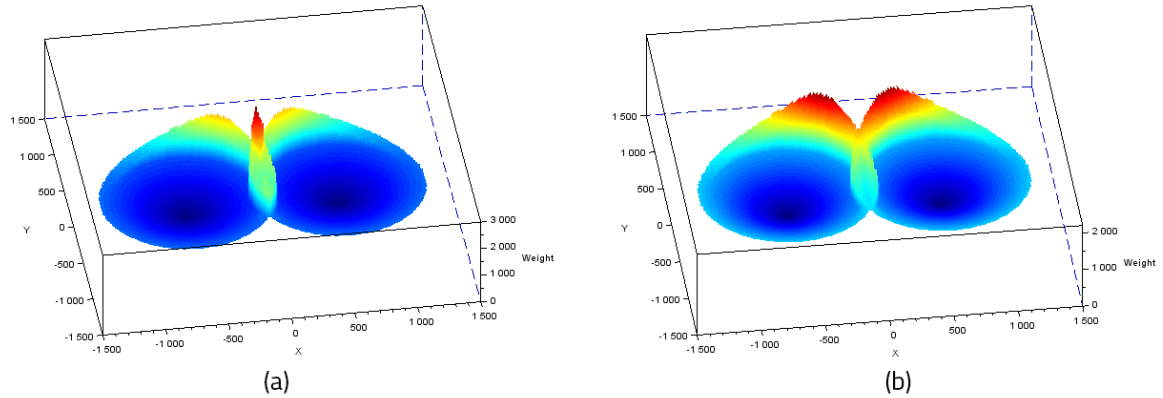

Figura 4.5. *(a)* Abordagem utilizando o método de soma. É possível identificar claramente um ótimo local, note que o local é mínimo, não permite movimentação e possui um peso alto; *(b)* Abordagem utilizando o método de seleção do maior valor. O ótimo local desaparece e há uma melhor distribuição de peso, permitindo ao herói se movimentar. Em ambas as plotagens vermelho é mais desejável e azul menos desejável.

4.1.7 Heróis

Os heróis são as unidades mais dinâmicas encontradas em um jogo MOBA. Essas unidades são mais poderosas do que qualquer outra encontrada no jogo, sendo comumente controlada por um jogador humano. O herói inicia a partida no nível 1 e deve desenvolver os seus níveis, melhorar suas habilidades e características, comprar itens, coletar dinheiro e executar habilidades. A maioria dos MOBAs possui uma grande variedade de heróis disponíveis, League of Legends por exemplo, possui mais de 120 heróis disponíveis para ser utilizados.

O agente desenvolvido nesta pesquisa considera heróis aliados e heróis inimigos durante o processo de análise tática. É importante observar que nesta abordagem são

[2]Ótimo local, no contexto de mapa de influências, é um local composto por uma ou mais células que recebem um alto valor de influência, fazendo com que o agente fique preso nesta área.

desconsiderados os poderes específicos do herói, uma vez que se tem por objetivo fazer com que o agente possa ser o mais generalista possível.

Heróis aliados são aqueles que fazem parte do mesmo time que o agente. Estes heróis não podem infligir dano ao agente, todavia, podem conceder habilidades ou *status* através de habilidades específicas ou mesmo colidir com o herói. Normalmente, os heróis aliados cooperam entre si com o objetivo de conquistar a base inimiga destruindo o Nexus. Como discutido, no modo mais comum de jogo, cada time é composto por cinco heróis. Todavia, há modos de jogo que permitem partidas de três jogadores em cada time ou mesmo um jogador contra outro jogador.

O agente considera os heróis aliados como uma força atrativa, significando que há uma recompensa em se posicionar próximo a estes heróis. Todavia, devido à dinamicidade destes heróis, deve-se considerar manter alguma distância, a fim evitar a colisão com os mesmos, de forma similar ao que acontece com as torres aliadas. Portanto, há uma pequena força repulsiva em volta de cada herói aliado.

Heróis inimigos são aqueles que se encontram no time oposto ao do agente. Estes heróis podem infligir dano ao agente e a seus aliados. Portanto, considera-se que estes inimigos geram uma força repulsiva, significando que não é uma boa prática se aproximar demais dos mesmos. Os heróis inimigos podem ainda gerar uma força atrativa caso o agente tome a decisão de atacá-lo em caso de certeza de que pode derrotá-lo.

4.2 Um agente de suporte ao jogador em League of Legends

Os jogos do gênero MOBA são conhecidos por sua grande complexidade e dificuldade de aprendizado. Novos jogadores, normalmente, têm dificuldade em se adaptar ao jogo, devido à sua jogabilidade única e grande quantidade de informações a serem manipuladas. Além disso, jogos MOBA são cooperativos, exigindo o contato de um jogador iniciante com outros jogadores. Este contato nem sempre é amigável, gerando os comportamentos tóxicos[3] [Pobiedina et al., 2013].

Tendo em vista os problemas citados, propõe-se então um agente de suporte ao jogador. Este agente, além de dar dicas, joga partidas juntamente ao jogador substituindo um eventual jogador tóxico. O agente ainda é capaz de auxiliar o jogador

[3]São considerados comportamentos tóxicos aquelas ações que provêm de um jogador e podem, direta ou indiretamente prejudicar outros jogadores. Exemplos de comportamentos tóxicos são: linguagem ofensiva, roubo, jogadas que influenciam negativamente outros jogadores, dentre outros.

em sua sobrevivência, utilizando um herói fortemente voltado para o papel de suporte, mitigando assim a frustração inicial encontrada no gênero. Como plataforma de testes, utiliza-se o jogo LoL, selecionando o herói *Soraka* como suporte.

4.2.1 O Herói: Soraka

Fazer com que um jogador iniciante possa aprender MOBA inclui vários fatores, mas o principal deles, sem dúvida, é o herói. Em League of Legends, por exemplo, jogadores iniciantes possuem um conjunto específico de heróis que são considerados "fáceis", até que atinjam determinado nível. Selecionar heróis complexos logo no início da experiência do jogo pode ser extremamente frustrante, pois dificilmente o jogador se sairá bem.

Outro fator impactante nos primeiros jogos são os demais jogadores humanos com o qual o jogador terá de cooperar. Se o comportamento destes jogadores for tóxico, pode ser que o jogador ache o ambiente de jogo hostil e desista de jogar. Para tentar mitigar estes problemas, substitui-se um do jogadores por um agente inteligente, sempre no controle do herói Soraka.

Soraka é um herói que normalmente assume o papel de suporte. Esta função tem por responsabilidade principal ajudar o time concedendo bênçãos e magias de auxílio. Além disso, o suporte é responsável por ajudar o jogador na posição de Carregador a se fortalecer, jogando na rota inferior. Selecionou-se então esta função para que o agente pudesse a preencher dentro do jogo.

O herói Soraka foi escolhido após se analisar todos os heróis que foram desenvolvidos com o objetivo de assumir o papel de suporte. Soraka se mostrou o herói mais adequado por possuir uma variada gama de habilidades que, ao mesmo tempo, auxiliam na sobrevivência do jogador e o auxiliam a derrotar inimigos. Soraka pode ser vista na Figura 4.6.

Soraka[4] possui quatro habilidades principais e uma habilidade passiva. A habilidade passiva se chama Salvação, e permite ao herói se mover mais rapidamente em direção aos aliados feridos. Isto permite ao agente ser mais ativo durante a jogabilidade não só do jogador alvo, como a de múltiplos jogadores. Sua habilidade primária é denominada Chamado Estelar, e quando utilizada invoca um projétil que atrasa e causa danos aos inimigos na área. Além disso, caso o projétil atinja algum inimigo, Soraka é curada. Sua habilidade secundária é denominada Infusão Astral, e cura um aliado quando utilizada, doando o HP de Soraka. A terceira habilidade, denominada Equinócio, cria uma área onde os inimigos são silenciados, tornando-os incapazes de

[4]Mais informações sobre o herói podem ser encontradas em sua página: http://riot.com/1OTMKbi

Figura 4.6. Soraka e suas habilidades dentro do contexto de League of Legends. Na parte superior esquerda observa-se a habilidade *Chamado estelar*. Na parte superior direita observa-se a habilidade *Desejo*. Na parte inferior esquerda, observa-se uma visão do herói em sua versão Ceifadora, uma personalização oferecida no jogo. Na parte inferior central observa-se a habilidade *Equinócio*. Na parte inferior direita observa-se a habilidade Infusão Astral.

lançar feitiços ou habilidades. Após um curto período de tempo, a área de equinócio é destruída, e inimigos que se encontrem dentro da área são enraizados, tornando-os incapazes de se movimentar. A última habilidade de Soraka é chamada de Desejo, e cura todos os aliados vivos no campo de batalha.

Selecionou-se então o herói devido às suas habilidades e características, a fim de se implementar o sistema de suporte. Além disso, Soraka é capaz de auxiliar múltiplos jogadores, e não somente o jogador aliado, auxiliando não somente um jogador, mas todo o time a evitar a frustração e melhorar suas habilidades.

4.2.2 Sistema de Suporte ao Jogador

Após a seleção de um Herói em específico para a implementação do agente, foi iniciada a implementação do sistema. Inicialmente implementou-se uma abordagem que utilizava FSM, porém esta técnica tornou-se improdutiva, dada a complexidade do agente. Posteriormente foi implementada uma abordagem baseada em GOAP, sendo que esta demonstrou-se mais facilmente escalável que a técnica de FSM. Todavia, o agente nem sempre executava as mesmas ações. Testes preliminares com usuários demonstraram

que o agente se comportava de forma não contínua, o que causava estranhamento ao usuário. Acredita-se que este estranhamento seja devido à falta de padrões de comportamento do agente, previamente estudada por Koster [2013] em sua teoria da diversão.

Identificada a necessidade de escalabilidade e de um comportamento estático, decidiu-se por reimplementar o agente em uma estrutura que pudesse atender ambas as necessidades. Decidiu-se então implementar uma Árvore de Comportamento, implementando comportamentos estáticos para o agente. Para gerar estes comportamentos, foi feita uma pesquisa que envolveu a análise de *replays*, visualização de *livestreams* de jogadores profissionais e *feedback* de jogadores de alto nível. A partir desta pesquisa, extraiu-se um conjunto de comportamentos essenciais a um jogador que queira atuar como suporte. Estes comportamentos foram então implementados em uma Árvore de Comportamento que pode ser visualizada na Figura 4.7. A Árvore de comportamento é executada a cada *frame* de jogo, sendo responsável por executar ações utilizando o herói. Os seguintes comportamentos são implementados pela árvore:

- Execução após determinado limite de tempo inicial do jogo;
- Verificação de seleção de um jogador parceiro;
- Verificação de atividade do jogador parceiro;
- Verificação de distância entre o herói e o parceiro;
- Verificação de distância entre o herói e jogadores aliados;
- Seleção de jogador parceiro;
- Verificação de dano recebido por torres;
- Evasão de dano causado por torres;
- Verificação e execução de volta à base aliada;
- Seguir o jogador parceiro;
- Compra de itens;
- Volta segura em caso de morte do parceiro.

Após a implementação, o agente passa a coletar dados sobre o ambiente para que possa se comportar de forma racional, com o objetivo de auxiliar o jogador. Estes dados alimentam então a Árvore de Comportamento e causam estímulos ao agente,

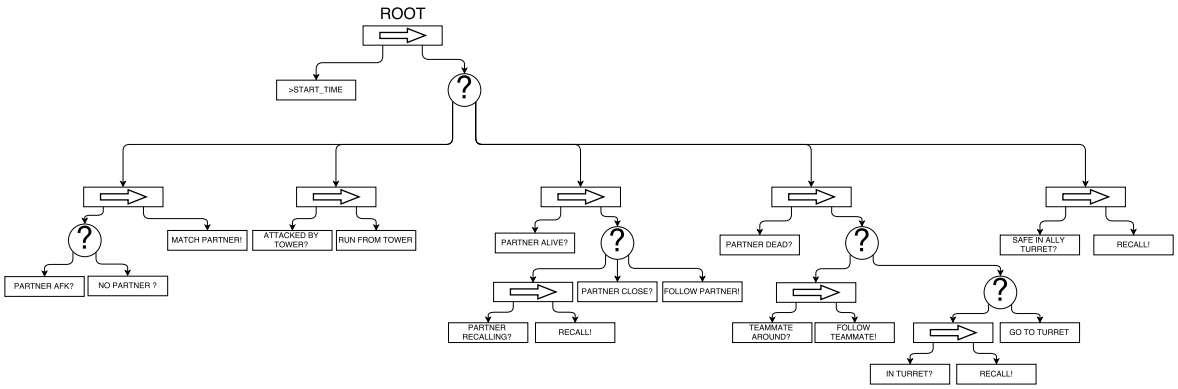

Figura 4.7. Árvore de Comportamento implementada no agente de suporte ao jogador.

que reage a estes dados. Como saída, o agente deve realizar uma ação, previamente implementada na forma de um nó de ação.

Em suma, o agente é inserido no jogo juntamente com outros quatro jogadores humanos. Como este agente foi desenvolvido para jogar partidas com jogadores iniciantes, os jogadores inimigos são IAs do próprio jogo, implementadas pela produtora. Ao iniciar o jogo, o agente seleciona um jogador humano como parceiro, o qual seguirá e dará suporte em tempo integral. Por outro lado, dará suporte aos demais jogadores em tempo parcial, quando possível.

4.2.3 Sistema de Dicas

O sistema de dicas foi implementado com o objetivo de auxiliar os jogadores através de dicas escritas na caixa de diálogo do jogo. Estas dicas seriam o equivalente ao que um jogador faz ao escrever orientações para o jogador iniciante. Modelou-se este sistema como um *Sistema Baseado em Regras*, onde cada regra possui um conjunto de dicas que podem ser emitidas para o jogador.

Como entrada, o sistema lê as informações disponíveis em tempo real no jogo e também as informações disponíveis no mapa de influência anteriormente discutido neste trabalho. Analisando estas informações, o sistema verifica qual regra será ativada, retornando uma dica equivalente à regra ativada.

Além da implementação básica, foi implementado um controle de tempo de recarga para a emissão de regras, a fim de não causar um grande fluxo de informação no *chat* de jogo. Ou seja, a regra tem um tempo de espera para ser ativada novamente, emitindo mensagens quando ativada apenas se o intervalo definido houver sido satisfeito. Outra preocupação é a de que o sistema não fosse monótono, emitindo sempre

a mesma frase. Para solucionar este problema foram redigidas diferentes frases com o mesmo significado, a fim de fazer com que as dicas não fossem repetitivas. Estas mensagens eram atribuídas a uma regra e selecionadas aleatoriamente durante a ativação da mesma. Um exemplo de exibição de mensagem pode ser observado na Figura 4.8. A tabela completa de regras, dicas e intervalos pode ser visualizada na Tabela B.1.

Figura 4.8. Exemplo de interação do sistema com um jogador. O nome dos jogadores foi omitido a fim de se manter a privacidade.

As regras implementadas são capazes de dar dicas para as seguintes situações:

- dano causado por herói inimigo;
- dano causado por torre inimiga;
- dano causado por tropas inimigas;
- HP baixo;
- alto uso de recursos;
- falhas na mecânica de *farming*;
- posicionamento inadequado;
- possibilidade de intimidação do inimigo;

- possibilidade de derrotar um herói inimigo.

A detecção de cada uma destas situações permite ao sistema emitir uma mensagem ao jogador a fim de auxiliá-lo. Por exemplo, ao detectar baixo HP, o sistema pode emitir uma mensagem para o jogador utilizar uma poção ou retornar à base aliada. A detecção de posicionamento utiliza o mapa de influência computado, a fim de informar ao jogador que aquele posicionamento é perigoso.

Além das dicas, o sistema é capaz de utilizar um sistema de *pings*, que são alertas inteligentes sem mensagem. Conjuntamente, este sistema fortalece o conceito estabelecido pelo sistema de dicas, auxiliando o jogador durante seu processo de aprendizado.

4.3 Sumário

Em resumo, foram desenvolvidos dois agentes inteligentes baseados em conhecimento para o ambiente MOBA. O primeiro agente, objetivo principal deste trabalho, foi concebido utilizando uma arquitetura heterogênea, sendo capaz de jogar partidas no ambiente de LoL por meio da modelagem de pesos do atributo conhecido como *aggro*. Com isto, este agente é capaz de realizar uma análise tática do ambiente, tomando decisões sobre movimentação e comportamento.

O segundo agente teve por objetivo auxiliar novos jogadores em suas partidas iniciais de LoL. Este auxílio acontece jogando juntamente a outro jogador na posição de suporte. Além disso, o agente é capaz de analisar o ambiente de jogo por meio do arcabouço tático proposto e prover dicas ao jogador iniciante, tentando assim prover *feedback* para melhorar sua habilidade. A estrutura deste agente, no entanto, não é baseada na análise tática, mas antes em conhecimento modelado em uma Árvore de Comportamento, técnica escolhida por seu comportamento sistemático.

Capítulo 5

Experimentos

Neste capítulo, são apresentados os experimentos executados para a validação dos agentes desenvolvidos durante este trabalho. Apresenta-se inicialmente uma descrição do ambiente de teste de forma geral e da métrica de *KDA*, conhecimentos essenciais para a compreensão dos experimentos. Ainda no contexto introdutório, apresenta-se as ferramentas e métodos de coletas de dados utilizados.

Segue-se então para a discussão dos experimentos para validação do agente capaz de jogar MOBA, apresentado na Seção 4.1.1. Posteriormente, apresenta-se os experimentos executados para a validação do agente de suporte aos novos jogadores, apresentado na Seção 4.2.

5.1 Preâmbulo

Os testes descritos neste capítulo foram realizados utilizando um ambiente MOBA específico: o jogo League of Legends. Neste jogo, como discutido anteriormente, não há uma plataforma de programação nativa, o que requeriu a utilização de ferramentas de terceiros. Além disso, o jogo possui terminologias próprias e métricas herdadas de outras áreas, como *eSports*. Esta seção é dedicada a discutir a coleta de dados e métricas utilizados nos experimentos realizados.

5.1.1 O ambiente de testes

O ambiente de testes utilizado nos experimentos foi o jogo League of Legends, mais especificamente dois mapas: *Howling Abyss* e *Summoner's Rift*. Nestes mapas é possível inserir de um a cinco jogadores em cada time. Além disso, é importante observar que

o jogo possui uma IA nativa implementada, apesar de ela só se encontrar disponível no mapa *Summoner's Rift*.

O ambiente utilizado é exatamente o mesmo utilizado em jogos competitivos e casuais pelos jogadores, garantindo assim que o ambiente seja familiar aos jogadores que possam participar dos testes. Além disso, a utilização do ambiente nativo de jogo possibilita a reprodução dos experimentos dentro dele.

Para a execução dos códigos dentro do jogo foi utilizada a ferramenta Bot of Legends (BoL), que permite a execução de códigos na linguagem Lua. Esta ferramenta permite ao desenvolvedor executar funções nativas ou desenvolver funções próprias. A mesma apenas oferece ao desenvolvedor informações que estejam disponíveis ao jogador, garantindo assim que o mesmo não se utilize de trapaças. Todos os códigos foram desenvolvidos na linguagem Lua e podem ser executados utilizando a plataforma BoL. Os mesmos códigos estão disponíveis na plataforma *GitHub* do autor[1].

5.1.2 Métrica KDA

Para mensurar o desempenho de jogadores, foi necessária a criação de uma métrica que medisse o desempenho dos mesmos dentro do jogo. Para tanto, a métrica mais comum é medir o número de Abates, Mortes e Assistências (do inglês, *Kills*, *Deaths and Assistances*), ou simplesmente *KDA*. Esta métrica é largamente utilizada no meio competitivo do jogo, sendo utilizada para apontar os melhores jogadores. Um bom jogador normalmente possui um *KDA* estável, sendo o mesmo variável conforme a posição que o jogador assume dentro do jogo.

Normalmente, um valor alto de *KDA* quer dizer que o jogador foi altamente participativo durante uma partida. Por outro lado, um jogador que possuiu uma participação menor terá um *KDA* menor. Os elementos isolados que compõe esse fator também são levados em conta em relação ao desempenho dentro da partida. Um valor elevado de K (Abates) indica que o jogador coletou a maioria dos abates, comportamento normalmente assumido por jogadores de heróis que tendem a causar grande impacto na partida, os chamados carregadores. Um jogador que tenha um valor elevado de D (Mortes) normalmente é aquele que tem o pior desempenho, uma vez que recebe maiores punições de tempo e concede dinheiro e experiência ao time inimigo. Finalmente, jogadores que obtêm elevados valores de A (Assistências), possuem participação ativa ou passiva nos abates. Os jogadores com maior pontuação A normalmente são os suportes e os *junglers*, uma vez que estes tem a função de auxiliar o time na execução de jogadas que possam resultar em abates de campeões inimigos.

[1] https://goo.gl/d2YY4u

O KDA pode ser calculado através de uma fórmula específica, considerando seus fatores isoladamente. O cálculo do fator é demonstrado na Equação 5.1.

$$KDA = \begin{cases} \frac{K+A}{D}, & D > 0 \\ K + A, & \text{senão} \end{cases} \tag{5.1}$$

Em resumo, os jogadores procuram otimizar o fator *KDA* obtido nas partidas por meio da maximização dos valores de K e A, enquanto minimizam o valor de D. Todavia, este comportamento nem sempre é o presente nas partidas. Há por exemplo estratégias que requerem o sacrifício de um jogador a fim de obter a vantagem de posicionamento e então o time aliado tomar a vantagem. Assim, o fator *KDA* não deve ser tomado como um indicador isolado de bom desempenho nas partidas, uma vez que os eventos acontecidos na mesma partida também possuem caráter decisivo frente as estratégias adotadas pelo time. Portanto, em alguns testes, são adotadas métricas adicionais, a fim de verificar o desempenho do agente em diferentes cenários de teste.

5.1.3 Coleta de dados

O ambiente de League of Legends oferece ao jogador uma série de estatísticas que podem ser acessadas no menu exibido após o jogo. Utiliza-se das estatísticas presentes nesta tela a fim de se registrar e mensurar o desempenho obtido pelo agente durante as partidas. Nesta tela é possível visualizar informações básicas, como a duração das partidas e o desempenho em Abates, Mortes e Assistências, bem como dados avançados da partida, como dano específico causado e recebido por cada unidade, dinheiro recebido, tropas, itens, dentre outros.

Além dos dados básicos e avançados informados pela partida, o jogo disponibiliza também uma ferramenta *online* de análise estratégica e de estatísticas, mostrando a progressão da equipe ao longo do tempo. Com esta ferramenta, é possível verificar quantitativa e qualitativamente o desempenho dos times ao longo da partida. Além disso, a ferramenta permite ao usuário filtrar os dados, exibindo características específicas de um ou mais jogadores, permitindo a análise de diversos aspectos da partida. Finalmente, a ferramenta exibe um mapa com abates e conquistas de estruturas que tenham acontecido durante uma partida. Uma visão parcial desta ferramenta de análise é mostrada na Figura 5.1.

Ambas as ferramentas foram utilizadas para análise do desempenho do agente e a coleta de dados necessários para as métricas propostas por cada um dos experimentos. Adicionalmente, utilizou-se a interface de programação para monitoramento em tempo real do agente, verificando seu funcionamento durante as partidas e para a verificação

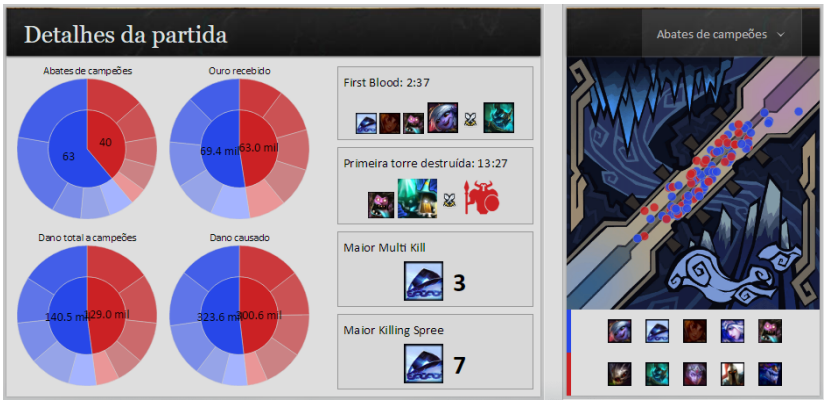

Figura 5.1. Visão parcial da ferramenta de análise de partidas de LoL. A esquerda observa-se projeção visual do desempenho de equipes. A esquerda um mapa dos abates acontecidos ao longo da partida.

de ótimos locais.

5.2 Validação do Agente Capaz de Jogar MOBA

Discute-se nesta seção os testes realizados com os dois agentes desenvolvidos ao longo desta pesquisa. Apresenta-se primeiramente os testes realizados com o agente desenvolvido a fim de jogar MOBA. O primeiro experimento trata sobre a eficácia do agente em interagir com o ambiente e vencer partidas de MOBA. O segundo experimento testa a eficiência do agente em relação ao *farming*. O último experimento trata da competição entre o agente desenvolvido e jogadores humanos.

5.2.1 Experimento 1 - Eficácia do Agente Capaz de Jogar MOBA

Neste experimento verificou-se a eficácia do agente em interagir e vencer partidas num ambiente MOBA. Para tanto, o agente deve ser capaz de coletar e processar informações do ambiente de forma a tomar decisões e atuar no ambiente a fim de vencer o jogo.

O teste consistiu em inserir o agente em um ambiente composto apenas de tropas e das estruturas defensivas, como torres inibidores e Nexus. Para tanto, utilizou-se o ambiente do jogo League of Legends. Devido à virtude do teste em vencer o jogo, selecionou-se um mapa composto apenas de uma rota, denominado *Howling Abyss*. Neste mapa cada time possui quatro torres, um inibidor e um Nexus. Além disso, em intervalos de tempo determinados há o surgimento de tropas em ambas as bases. Estas tropas vão em direção à base inimiga, lutando contra as tropas inimigas, torres, inibidores e capturando o Nexus. O objetivo do mapa é o mesmo: o time que capturar

o Nexus inimigo vence o jogo. Uma vista superior do mapa é apresentada na Figura 5.2

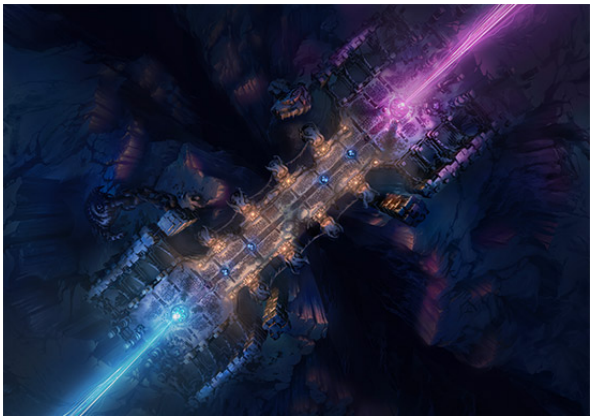

Figura 5.2. O mapa de *Howling Abyss*, ambiente de testes selecionado para o Experimento 1. Note que o mapa possui apenas uma rota.

Espera-se que durante este experimento o agente demonstre um comportamento racional, vencendo as tropas inimigas e derrotando as estruturas, sempre vencendo o jogo. Para tanto monitora-se o número de partidas vencidas. Utiliza-se ainda como métrica o tempo de duração da partida, a fim de verificar o tempo levado pelo agente para vencer o jogo com diferentes heróis. Durante estes testes não é utilizado o *KDA*, uma vez que não há interação com outros heróis, limitando a métrica apenas ao número de mortes sofridas pelo herói.

Para efeito de comparação, este experimento foi configurado o mais próximo possível do elaborado em Willich [2015]. Todavia, as abordagens utilizadas são diferentes: o trabalho de Willich [2015] utiliza a abordagem de Aprendizado por Reforço como principal mecanismo de controle do agente. Devido ao fato da técnica aplicada neste trabalho ser baseada em conhecimento e a técnica aplicada por Willich [2015] ser diferente, os efeitos comparativos são realmente divergentes. Enquanto a técnica de aprendizado por reforço requer tempo para o aprendizado e também uma série de ajustes em seus parâmetros, a técnica baseada em conhecimento requer apenas a codificação de conhecimento do ambiente para seu funcionamento. Portanto, na falta de um *baseline* compara-se o trabalho aqui desenvolvido com o desenvolvido por Willich [2015].

Foram executadas 20 partidas utilizando heróis aleatórios a partir da seleção de heróis disponível em League of Legends. O Relatório de partidas pode ser encontrado na Tabela A.1.

O agente se demonstrou eficaz e racional no ambiente MOBA, sendo capaz de

vencer todas as partidas. Isto significa que o agente foi capaz de iniciar o jogo, lutar contra tropas, destruir torres e inibidores e, finalmente, destruir o Nexus inimigo. O tempo médio de execução destas tarefas foi de 20 minutos e 28 segundos, com um desvio padrão de 3 minutos e 22 segundos. A média de mortes foi de 0.55 morte por partida. Sendo que 55% (11) das partidas não apresentaram nenhuma morte, 35%(7) das partidas apresentaram uma morte, e 10% (2) das partidas apresentaram duas mortes. Além dos dados apresentados, houve a coleta de movimentação do agente durante a partida, verificando que em nenhum momento o agente ficou estagnado em um ótimo local.

É possível observar que o agente se comportou de forma esperada, uma vez que venceu todas as partidas executadas. É interessante observar que, apesar de ser abatido, o agente é capaz de vencer as partidas. Em alguns casos, o agente consegue exibir uma maior velocidade de jogo após uma morte, como o caso da execução 4 e 15. Isto se deve a uma regra específica do mapa de *Howling Abyss*, onde só é possível comprar itens após sair da base ou sofrer um abate. Com isto, o agente pôde comprar itens poderosos ao ser abatido, acelerando o ritmo de jogo.

É importante observar ainda que o agente se mostra mais capaz de controlar diversas classes e diferentes naturezas de campeões, mesmo com campeões considerados fracos ou inadequados para o combate, como campeões da classe *suporte*. Apesar disso, o agente demonstra um melhor controle sobre campeões que possuem ataque à distância. Tanto em tempo de duração da partida quanto em relação ao número de mortes sofridas, o agente controla melhor heróis que atacam à distância do que heróis que atacam em curto alcance, também chamados de *melee*.

Tabela 5.1. Registro do tempo de duração médio da partida em relação a natureza do ataque do herói.

	Geral	Distância	*Melee*
Média	0:20:28	0:19:51	0:21:13
Desvio Padrão	0:03:22	0:04:18	0:02:23

Na Tabela 5.1 é possível observar o desempenho do agente quando controlando heróis de ataque à distância e *melee*. Há uma melhora de 37 segundos em relação à performance de heróis em geral quando o agente controla heróis de ataque à distância. Quando se compara os casos de ataque à distancia com o controle de heróis *melee*, o agente apresenta uma melhora de 1 minuto e 22 segundos no tempo médio de duração da partida. Por outro lado, ao controlar heróis de ataque à distância o agente demonstra uma maior instabilidade em relação à duração das partidas, atingindo um desvio padrão de 4 minutos e 18 segundos. Este desvio é justificável uma vez que, se analisados, os

heróis de ataque a distância possuem natureza muito diversa. Na execução 4, 13 e 15, por exemplo, os heróis possuem uma vantagem natural, devido ao ganho de status durante o *farming*, o que favorece o menor tempo de execução.

Tabela 5.2. Registro do número médio e desvio padrão do número de mortes sofridas pelo herói controlado pelo agente ao longo de 20 execuções.

	Geral	Distância	*Melee*
Média	0,550	0,455	0,667
Desvio Padrão	0,669	0,498	0,816

Ainda discutindo o melhor controle do agente sobre heróis de ataque à distância, é possível analisar o número de mortes sofridas pelos heróis controlados, resumidos na Tabela 5.2. Ao observar os dados disponíveis na Tabela A.1, é possível observar que todos casos onde o agente sofre mais de uma morte acontecem quando o agente estava no controle de herói *melee*. Isto se deve ao menor alcance de análise destes heróis, além do menor alcance de ataque, o que requer que o agente tome um maior risco ao executar mecânicas de *farming* ou de dano às torres (*sieging*).

O maior risco assumido ao controlar os heróis *melee* refletem diretamente no desempenho do agente, resultando em um aumento de mais de 21% na média de mortes sofridas em relação à média geral dos casos do experimento. Além disso há uma maior instabilidade no desempenho, atingindo um desvio de 0,816 morte por partida. Isto mostra que o agente tem menor expertise de controle de heróis *melee*, correndo mais riscos e, consequentemente, demonstrando um desempenho pior.

No geral, o agente mostra o desempenho esperado, uma vez que, mesmo sendo abatido, consegue se recuperar e vencer o jogo. O mesmo ainda não se detêm em ótimos locais e interage com o ambiente de forma racional. O tempo de partida se mostrou constante, mostrando que o agente é capaz de controlar diversos tipos de campeões e vencer partidas no ambiente MOBA, mesmo com campeões menos favorecidos para a tarefa designada.

5.2.2 Experimento 2 - Coleta de Recursos

Neste experimento, buscou-se validar a eficiência do agente na execução da mecânica de *farming*. Esta mecânica consistia em se aproximar de tropas e executar o último golpe, recebendo mais outro e experiência. Selecionou-se a mecânica de *farming* devido à sua importância na coleta de recursos para um melhor progresso do herói ao longo do jogo. Espera-se que o agente execute a mecânica de forma correta, coletando automaticamente dinheiro e experiência ao abater tropas.

Tabela 5.3. Desempenho do agente durante os testes de coleta de recursos. São apresentados dados de duração média da partida, número de tropas abatidas durante uma partida e número de tropas derrotadas por minuto.

Método	Duração (Média)	Tropas Abatidas (Média)	Tropas por Minuto
Jogador Profissional	-	-	10
[Willich, 2015]	-	-	3.333
ϕ desativado	00:31:52	194	6.085
ϕ ativado	00:30:15	279	9.226

Este experimento é conduzido em um mapa composto de três rotas dentro do ambiente de League of Legends, denominado *Summoners' Rift*. Foi necessária a utilização deste mapa uma vez que um dos objetivos deste experimento era também testar a eficiência do agente frente a outros agentes inteligentes. Todavia, em LoL não existia a implementação de uma IA nativa no mapa de *Howling Abyss*, o que impediu a utilização deste mapa. Portanto, assim como encontrado no experimento de [Willich, 2015], utilizou-se apenas a rota do meio. Foi inserido um agente inteligente controlado pela IA nativa do jogo para que este pudesse competir com o agente aqui implementado.

Para validação do teste utiliza-se a coleta de dados pós-jogo, compreendendo dados como campeões, classes, natureza do ataque, mortes e abates, duração das partidas e quantidade de tropas abatidas. Não foi necessária a coleta dos dados de assistências, uma vez que não há campeões aliados presentes na partida. Foram executados 20 partidas com heróis aleatórios, sendo cinco de ataque à distância e cinco *melee*. Dividiu-se a execução destas partidas em dois subconjuntos: o primeiro utiliza a Equação 4.5, onde a variável ϕ, responsável pela análise do HP das tropas, não está presente; o segundo utiliza a Equação 4.6, onde a variável ϕ analisa o HP das tropas inimigas. Para efeito comparativo, os heróis utilizados no segundo conjunto foram selecionados manualmente para replicar os heróis utilizados no primeiro experimento. Os dados coletados foram registrados e se encontram disponíveis nas Tabelas A.2 e A.3 respectivamente.

Além do *baseline* provido por [Willich, 2015], decidiu-se comparar o desempenho do agente com o desempenho de jogadores profissionais de MOBA. Existe um senso comum da área de *e-Sports* que diz que, no ambiente de LoL, um jogador excelente consegue coletar cerca de 10 tropas por minuto. Utiliza-se este conceito como *baseline* de eficiência máxima para efeito de comparação, neste experimento.

Na Tabela 5.3 apresenta-se um resumo dos dados disponíveis. Observou-se que, quando a Equação 4.5 foi utilizada, o agente apresentou uma eficiência de 60.84% em relação à *performance* de jogadores profissionais. Já quando a Equação 4.6, onde ϕ é ativada, o agente apresentou uma eficiência de 92.24% em coleta de recursos, uma melhora de mais de 30%. Pode-se concluir então que a análise do HP, por meio da variável ϕ é importante durante a análise de tropas para o *farming*.

5.3 Auxiliando Novos Jogadores

Discute-se nesta seção os experimentos realizados com o agente desenvolvido com o intuito de auxiliar jogadores iniciantes de League of Legends. O primeiro experimento consistiu em fazer com que o agente jogasse juntamente a jogadores aleatórios selecionados pelo sistema de gerenciamento de partidas. Já o segundo experimento, consistiu em testar o agente em ambiente controlado juntamente a jogadores selecionados. Neste segundo experimento, além de jogar partidas, os jogadores responderam questionários, a fim de avaliar o agente.

5.3.1 Experimento 1 - Auxiliando Jogadores Designados pelo Sistema de Gerenciamento Partidas

Este experimento teve por objetivo avaliar o agente quanto à sua capacidade de auxiliar novos jogadores em suas primeiras partidas de LoL. Para tanto utilizou-se um sistema de gerenciamento de partidas disponível no jogo. Esperou-se que o agente pudesse auxiliar jogadores iniciantes em suas partidas, fazendo-os aprender o jogo mais facilmente e oferecendo uma experiência mais agradável, prevenindo o comportamento tóxico que possa acontecer.

O experimento foi conduzido introduzindo-se o agente em partidas contra a IA nativa do jogo juntamente a outros jogadores humanos aliados. Para tanto foi criada uma nova conta, uma vez que o sistema seleciona jogadores de nível parecido. O ambiente foi o mapa de *Summoner's Rift*, sendo um time composto por quatro humanos selecionados pelo sistema do jogo e o agente de suporte, e o time adversário composto por cinco heróis controlados pela IA nativa do jogo. Os humanos envolvidos não foram informados de que estariam jogando juntamente com uma IA. Os testes foram divididos em dois conjuntos: a) três partidas apenas com o sistema de suporte ao jogador; b) três partidas com o sistema de suporte ao jogador e o sistema de dicas via *chat*.

Neste experimento a principal métrica foi o desempenho em *KDA* do jogador o qual o agente seleciona como parceiro. Para tanto coletou-se o histórico de partidas do jogador parceiro, sendo analisados os dados das três partidas anteriores, os dados da partida jogada juntamente ao agente, e os dados das três partidas subsequentes.

Para preservar a identidade dos jogadores envolvidos no experimentos, substituiu-se seus nomes por letras de A a F. Os jogadores A, B e C tiveram o auxílio apenas do sistema de suporte ao jogador, já os jogadores D, E e F tiveram suporte do sistema de dicas e do sistema de suporte. Executou-se esta separação a fim de evidenciar a diferença de desempenho entre os jogadores que tivessem suporte do sistema de dicas e

dos que tiveram suporte sem o sistema de dicas. A Figura 5.3 mostra uma interpretação visual dos dados coletados.

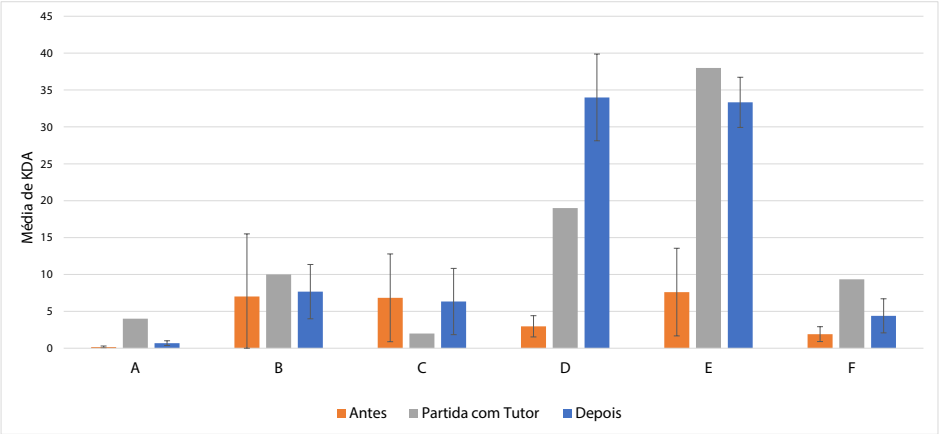

Figura 5.3. Gráfico que mostra o desempenho dos jogadores A a F nas partidas. A barra laranja mostram as partidas antes da experiência com o agente; A barra cinza mostra a partida realizada com o agente; A barra azul mostra as três partidas subsequentes a partida com o agente.

Analisando os dados coletados é possível notar que todos os jogadores, exceto C, conseguiram um melhor resultado quando jogaram com o agente de suporte do que nas três partidas anteriores. Este resultado indica que o agente pode ter sido um melhor parceiro para estes jogadores do que seus aliados anteriores. É interessante ainda observar que os jogadores D, E e F possuíam o sistema de dicas ativado durante sua partida juntamente ao agente de suporte. Estes jogadores também tiveram melhora de performance, tendo superado percentualmente e absolutamente todos os resultados das partidas anteriores.

Quando comparado às partidas posteriores à jogada juntamente ao agente de suporte, apenas o jogador D apresentou uma melhor performance. Isto indica que talvez não tenha acontecido o aprendizado da jogabilidade. Entre os motivos pelos quais este aprendizado pode não ter acontecido está o número de partidas jogadas. Possivelmente apenas uma partida não seja suficiente para o ensino das técnicas aos novos jogadores, e o jogador D pode ter sido uma exceção.

Ao comparar apenas as partidas anteriores e subsequentes à partida jogada juntamente ao agente de suporte, é interessante observar que todos os jogadores que tiveram o auxílio do agente juntamente ao sistema de dicas (D, E e F) mostraram progresso significativo em relação ao desempenho anterior. Já os jogadores que tiveram apenas o suporte do agente, sem o sistema de dicas (A, B, C), demonstraram um desempenho estável nas partidas posteriores em relação às anteriores. Isto indica que, o sistema

de dicas pode ter ajudado estes jogadores a identificar falhas em sua jogabilidade e corrigi-las.

Dentre os jogadores analisados, nota-se que o indivíduo C foi o único que demonstrou uma piora em seu desempenho, durante e após as partidas jogadas. Não se pode afirmar com certeza o que causou esta piora, porém é possível realizar especulações. Dentre as possibilidades técnicas, encontram-se: má conexão e falha de equipamentos. Também é importante ressaltar que o jogador é humano, e encontra-se propenso a vários fatores, como distrações, cansaço, níveis de proficiência, dentre outros.

Para analisar a consistência da melhora de jogabilidade, mediu-se o desvio padrão do *KDA* dos jogadores. Verificou-se que anteriormente à partida do agente de suporte com o jogador, os jogadores apresentavam um alto desvio padrão de *KDA*, fato que pode ser considerado aceitável para jogadores iniciantes. Os resultados demonstram que os jogadores B, C e E reduziram o desvio padrão após jogar juntamente ao agente, demonstrando uma maior estabilidade de jogabilidade. Por outro lado, os jogadores A, D e E demonstraram um maior desvio padrão em seu desempenho, demonstrando sua maior instabilidade durante o jogo. Relacionando estes dados ao desempenho melhorado, é possível que os jogadores que mais aprenderam com o agente são os que mais melhoraram seu desempenho. Porém, apesar do aprendizado, ainda se demonstram instáveis enquanto jogadores.

Além dos resultados, verificou-se por meio do sistema de honras disponível em League of Legends que o agente recebeu honras dos demais jogadores. Após seis partidas, o agente recebeu três honras, sendo duas por ser prestativo e uma por trabalho em equipe. O agente recebeu ainda uma solicitação de amizade e alguns elogios por sua postura via *chat*.

As dúvidas levantadas por este experimento questionavam vários aspectos da pesquisa, como a utilidade efetiva do agente e do sistema de dicas. Para validação destas questões foi necessária a realização de um novo experimento com maior controle, o qual é apresentado na seção a seguir.

5.3.2 Experimento 2 - Questionários e usuários voluntários

Este experimento teve por objetivo verificar a utilidade do agente para jogadores de *League of Legends*. Para tanto, um conjunto de seis jogadores voluntários foram convidados a participar deste experimento.

Os jogadores selecionados foram convidados a responder dois questionários, sendo o primeiro um questionário preliminar e o segundo um questionário após os experimentos. Além dos questionários, os jogadores foram convidados a jogar duas partidas de

LoL junto ao agente desenvolvido, sendo informados que estariam jogando juntamente a uma IA. O agente foi inserido em partidas gerenciadas pelo sistema do próprio jogo juntamente ao usuário convidado. O time aliado era composto portanto por três jogadores designados pelo sistema de gerenciamento de partidas, um jogador convidado e o agente. O time inimigo era composto por cinco heróis controlados pela IA nativa do jogo. Devido à necessidade da presença da IA, as partidas foram executadas no mapa de *Summoner's Rift*. Foi solicitado ao jogador convidado que assumisse o papel de parceiro do agente, garantindo assim que o agente o desse suporte ao longo da partida.

Inicialmente, cada jogador foi solicitado a responder o questionário preliminar, disponível no Apêndice C.1. Após concluída esta etapa, os jogadores jogaram duas partidas: a primeira acompanhado do agente de suporte com o sistema de dicas desativado; e a segunda partida acompanhado do agente de suporte com o sistema de dicas ativado. Após finalizar as duas partidas, cada jogador foi solicitado a responder o questionário após os experimentos, disponível no Apêndice C.2. Coletou-se os dados pós-jogo e dados dos questionários, criando-se fichas individuais dos jogadores. No total, seis jogadores realizaram os testes propostos. Os resultados são apresentados a seguir.

No total, seis jogadores se voluntariaram a realizar os testes. Inicialmente procurou-se traçar o perfil dos jogadores de teste, conforme mostra a Figura C.1. Estes jogadores são, em sua totalidade, do gênero masculino com idades entre 18 e 24 anos.

Após coletar os dados iniciais o questionário busca identificar o perfil de jogador do usuário, conforme mostra a Figura C.2. Esta identificação é importante para verificar a credibilidade e a assertividade das respostas dadas ao questionário. Todos os jogadores tinham experiência prévia com MOBAs e com LoL. Em relação à expertise, 50% (3) dos jogadores se identificam como avançados, 33% (2) se identificam como intermediários e 16% (1) se identifica como iniciante. Esta questão é chave para que se possa ter credibilidade em relação à confiança do jogador. Além disso perguntou-se o tempo de experiência com MOBAs, sendo que mais de 50% do jogadores tem de 1 a três anos de experiência com MOBAs.

Traçado um perfil geral do jogador, solicitou-se que o mesmo respondesse uma série de questões relativas ao seu perfil em *League of Legends*, uma vez que esta seria a plataforma de testes utilizada. As perguntas específicas sobre LoL se encontram na Figura C.3. Identificou-se com este questionário que 50% dos jogadores possuem a posição de *Atirador* como predileta.

Procurou-se identificar o conhecimento e experiência com o herói Soraka, uma vez que este seria utilizado nos testes. Quando questionados se conheciam o herói Soraka, 83% (5) dos usuários responderam que **sim**, enquanto 16% (1) responderam que não.

Questionados sobre sua experiência em jogo com Soraka, 67% (4) dos usuários responderam que já jogaram utilizando o herói Soraka, enquanto que 33% (2) responderam que nunca jogaram utilizando Soraka como herói.

Finalmente, identifica-se a experiência geral do usuário com o tutorial de League of Legends. Quando questionados se realizaram o tutorial ao começar a jogar LoL, 67% (4) dos usuários responderam que sim, fizeram o tutorial, enquanto que 33% (2) disseram nunca ter feito o tutorial de LoL. Após responder se havia feito o tutorial, foi solicitado ao usuário que avaliasse a suficiência do tutorial para o jogador iniciante. Todos os jogadores concordam que o tutorial não é suficiente para ensinar um jogador iniciante sobre LoL.

Finalizado o questionário preliminar, o usuário foi convidado a jogar duas partidas na companhia do agente de suporte ao jogador. Na primeira partida, o jogador foi acompanhado apenas do agente de suporte, sem o sistema de dicas. Já na segunda partida o jogador pôde experimentar o sistema de suporte aliado ao sistema de dicas. Após finalizar as duas partidas, procurou-se validar o agente de suporte ao jogador por meio de um novo questionário, os resultados obtidos são apresentados a seguir.

O questionário aplicado posteriormente aos testes procurou validar o agente, fazendo perguntas sobre a utilidade do mesmo para jogadores iniciantes na visão dos usuários. As perguntas aplicadas neste questionário se dividem em dois grupos: questões sobre a avaliação do agente (Figura C.4); e perguntas sobre o comportamento do agente de suporte (Figura C.5).

Inicialmente solicitou-se ao usuário que desse uma nota ao sistema, utilizando uma escala numérica horizontal com valores de 0 a 10, onde 0 significava muito ruim, e 10 significava muito bom. Todos os jogadores atribuíram o valor 10 à nota solicitada. Quando perguntados sobre a importância do sistema de dicas, todos os jogadores responderam que sim, o sistema de dicas é importante no agente de suporte. Ao serem questionados sobre quais mecânicas de jogo acreditavam que o agente poderiam auxiliar, todos os jogadores concordaram que o agente auxilia principalmente na mecânica de *farming*. Quando questionados sobre a utilidade das dicas para novos jogadores utilizando uma escala numérica de 1 a 10, os jogadores atribuem uma nota média de 8,5. Finalmente, questiona-se o jogador sobre a importância do sistema de dicas, 50% dos jogadores indicaram que o sistema possui a frequência ideal e 50% dos jogadores indicam que se requer um leve aumento da frequência.

Para finalizar o questionário posterior aos experimentos, questiona-se aos jogadores sobre o comportamento do agente durante o jogo. Inicialmente, solicita-se que se dê uma nota utilizando uma escala numérica de 0 a 10 para a característica de suporte do agente, onde 0 significa muito ruim, e 10 significa muito bom. A média obtida foi de 8.5.

Como última questão, pede-se que o jogador valide o comportamento do agente quanto à agir humanamente ou roboticamente. Nesta questão, todos dos jogadores concordam que o comportamento do agente foi mais robótico que humano. A característica robótica se deve ao comportamento estático apresentado pela BT implementada, e se encontra em concordância com o resultado esperado para este experimento.

Com a análise de tais respostas é possível afirmar que, segunda a avaliação dos usuários, o agente se comporta como esperado, auxiliando o jogador em seu aprendizado inicial. Além disso, verifica-se que os usuários avaliam que o sistema de suporte e de dicas são ferramentas válidas, apesar de necessitar de pequenos ajustes, e as consideram úteis para jogadores iniciantes. Este agente, segundo os usuários poderia ser utilizados pelos jogadores para aprender sobre mecânicas complexas, como o *farming*.

5.4 Sumário

Este capítulo discorreu sobre os testes validadores executados com os agentes desenvolvidos neste trabalho. Para tanto, apresentou-se o ambiente de testes, mapas e afins para melhor compreensão. Discutiu-se ainda a métrica *KDA*, largamente utilizada para medir o desempenho de jogadores em partidas de MOBA, tanto em ramos casuais quanto competitivos. Apresentou-se ainda o ambiente de coleta de dados disponibilizado por LoL.

Posteriormente, apresentou-se os experimentos com o agente de jogo em LoL. Nestes experimentos, validou-se primeiramente a eficácia e a eficiência do agente. A eficácia foi verificada ao inserir o agente dentro de um ambiente de jogo e permití-lo interagir com o mesmo, coletando dados sobre seu comportamento. Já a eficiência foi validada por meio da mecânica de *farming* mensurando o desempenho do mesmo em executar tropas adversárias.

Quando se observou os experimentos utilizados para validar o agente de suporte nota-se dois momentos. Primeiramente foi realizado um experimento preliminar com jogadores designados pelo sistema de gerenciamento de partidas do jogo. Então, prosseguiu-se para um experimento controlado, composto por questionários pré e pós-jogo, onde os jogadores foram convidados a jogar duas partidas. Estes questionários serviram para validar o conhecimento dos jogadores e, posteriormente, validar o agente em si.

Capítulo 6

Conclusão

Este trabalho apresentou e discutiu o desenvolvimento de agentes inteligentes para jogos MOBA. Foram apresentados aqui dois agentes: um agente que é capaz de jogar partidas de jogos MOBA e um agente capaz de auxiliar jogadores iniciantes deste gênero de jogo.

Para o desenvolvimento deste trabalho foi necessário primeiramente apresentar do que se trata o gênero, discorrendo sobre suas características básicas, passando por elementos de jogabilidade. Após esta apresentação procurou-se organizar e relacionar os trabalhos relacionados à pesquisa em MOBA. Todavia, um dos primeiros problemas enfrentados durante o desenvolvimento deste trabalho foi a escassez de pesquisa e bibliografia que utilizasse MOBA. Para preencher lacunas de pesquisa ainda não cobertas, utilizou-se trabalhos do gênero RTS, que possui desafios correlatos aos presentes em MOBA.

Após analisar as pesquisas já realizadas, procurou-se desenvolver um agente que solucionasse o problema de movimentação em MOBAs, primeiro tema deste trabalho. Entendeu-se então que os problemas de movimentação e combate estavam estritamente relacionados, e que a solução de um problema dependia fortemente da solução do outro. Isto se deve ao fato de o jogo permitir o combate entre seus personagens, fazendo com que o problema de combate modifique o problema de movimentação e vice-versa. Para tanto estendeu-se este trabalho e modificou-se o foco de pesquisa: ao invés de apenas pesquisar sobre movimentação procurou-se desenvolver um agente completo, que fosse capaz de tomar decisões no ambiente MOBA.

Para o desenvolvimento deste novo foco de trabalho foi necessária a pesquisa e implementação de técnicas que permitissem a modelagem do ambiente de forma simples e ao mesmo tempo eficaz. Para tanto identificou-se o parâmetro de *aggro*, que modela a agressividade presente em um dado espaço. Para a inserção deste parâmetro

sobre um mapa de jogo, procurou-se por técnicas que permitissem a análise e inserção de parâmetros, sendo mais tarde implementada a técnica de Mapas de Influência. Durante a implementação dos IMs, enfrentou-se vários problemas, como a manutenção de grandes estruturas de dados em tempo real, a seleção de parâmetros significativos para a análise e a mesclagem e simplificação de dados. Além disso, manutenções frequentes nos códigos eram necessárias, uma vez que o League of Legends recebe atualizações frequentes e a plataforma *Bot of Legends* se encontrava em constante desenvolvimento.

Ao longo do desenvolvimento deste trabalho foi ainda necessária a coleta de informações mais detalhadas sobre o jogo. Para tanto foi necessário o estudo de jogo, manuais, fóruns e técnicas. Além disso foi necessário aprender com jogadores profissionais, por meio de *livestreams* e tutoriais. Durante este trabalho de pesquisa virtual ,identificou-se que os jogadores iniciantes tinham dificuldade em aprender os conceitos básicos de LoL. Definiu-se então um objetivo secundário deste trabalho: o desenvolvimento de um agente que pudesse auxiliar novos jogadores.

O trabalho de desenvolver um agente que fornecesse suporte a outros jogadores é desafiador, uma vez que requer a interação com seres humanos. Mais do que isso, tem por objetivo não só de jogar com indivíduos com comportamento instável e imprevisível, bem como melhorar a técnica destes jogadores humanos. Para tanto, foram pesquisadas e implementadas diversas técnicas ao longo do desenvolvimento do agente de suporte ao jogador.

Inicialmente o agente de suporte foi implementado com uma Máquina de Estados Finitos. Logo no início do desenvolvimento percebeu-se que esta técnica se deprecaria rapidamente, uma vez que o número de estados crescia rapidamente conforme as necessidades identificadas. Migrou-se então o comportamento do agente para a técnica de Planejamento Baseado em Objetivos, que era substancialmente mais versátil porém apresentava uma desvantagem: seu comportamento não é estável. Devido à presença de comportamentos emergentes nesta técnica decidiu-se por parar de utilizá-la, uma vez que o jogador poderia ter estranhamento com comportamentos imprevisíveis. Finalmente implementou-se a técnica de Árvore de Comportamentos, que oferecia boa escalabilidade e comportamento estável, necessidades básicas do projeto.

Além do sistema de suporte ao jogador, foi implementado um Sistema Baseado em Regras capaz de dar dicas básicas sobre jogabilidade e mecânica ao jogador. Este sistema foi, mais tarde, integrado ao sistema de suporte. O objetivo principal deste era o de identificar e compensar falhas de habilidade do jogador por meio de dicas. Além disso o sistema avisava ao jogador sobre perigos iminentes e erros mecânicos no jogo.

Após a implementação de ambos os agentes, procurou-se validá-los. O primeiro foi validado no ambiente de jogo, sendo confrontado com o ambiente de jogo e com a IA

nativa implementada em LoL. O agente de suporte ao jogador foi validado utilizando jogadores humanos, que tiveram a oportunidade de jogar juntamente a ele e prover *feedback* por meio de questionários. Em ambos os casos concluiu-se que os agentes desenvolvidos cumpriram os objetivos para os quais foram desenvolvidos, uma vez que: i) o agente de jogo cumpre sua missão de conseguir jogar MOBA efetivamente; e ii) o agente de suporte consegue auxiliar jogadores e compensar falhas mecânicas.

Por fim, ao longo do desenvolvimento deste trabalho foram feitas contribuições, as quais são listadas a seguir:

- o uso de jogos MOBA como plataforma de testes para Inteligência Artificial, em especial League of Legends;
- uma arquitetura baseada em análise tática, como principal método de tomada decisão para agentes inteligentes em MOBA;
- implementação de um agente que é capaz de jogar e vencer partidas de LoL;
- emergência de um comportamento de *kiting*, baseando-se em mecanismos simples;
- implementação de um agente que é capaz de auxiliar novos jogadores a aprenderem a jogar League of Legends.

6.1 Trabalhos Futuros

Existem inúmeras possibilidades para a continuidade deste trabalho. Dentre elas, lista-se as seguintes:

- expansão da abordagem atual para uma abordagem multi-agente, utilizando um ou mais agentes para as mecânicas;
- desenvolvimento de um sistema especialista para cada herói, com o objetivo de extrair a melhor mecânica possível de cada um, como habilidades e *status*;
- aplicação de técnicas de Aprendizado de Máquina, como Aprendizado por Reforço, procurando aprender sobre o ambiente MOBA automaticamente;
- expansão do agente desenvolvido para suporte de habilidades;
- implementação de uma ferramenta de teste automático de balanceamento utilizando o agente capaz de jogar MOBA.

Uma das contribuições deste trabalho foi incentivar os pesquisadores a utilizarem MOBA como plataforma de testes em suas pesquisas. Acredita-se que este gênero provê um ambiente diversificado que pode ser utilizado por pesquisadores de diversas áreas, como: Interação Humano Computador, Inteligência Artificial, dentre outras. MOBA é, com certeza, um tópico de pesquisa interessante e que merece ser melhor estudado não só por sua popularidade, mas por sua larga abrangência de problemas e diversidade enquanto jogo e plataforma de teste.

Apêndice A

Tabelas de Execuções de Experimentos do Agente de Jogo em League of Legends

A.1 Tabela de Execução de Partidas para teste do Experimento 1

Tabela A.1. Execuções do Experimento 1 do Agente de Jogo MOBA. Lista-se as 20 execuções, o nome dos jogadores, bem como suas respectivas classes e natureza de ataque. A duração das partidas é descrita no formato hh:mm:ss. Registra-se ainda o número de Mortes ocorridas.

# Execução	Herói	Classe	Ataque	Duração	Mortes
1	Thresh	Suporte	*Melee*	0:21:25	0
2	Twisted Fate	Mago	Distância	0:20:30	1
3	Sona	Suporte	Distância	0:20:26	0
4	Jinx	Atirador	Distância	0:14:34	1
5	Wukong	Lutador	*Melee*	0:18:56	0
6	Ashe	Atirador	Distância	0:24:24	0
7	Jax	Lutador	*Melee*	0:19:58	1
8	Vayne	Atirador	Distância	0:18:35	1
9	Fiora	Lutador	*Melee*	0:18:57	0
10	Nautilus	Tanque	*Melee*	0:21:24	2
11	Ekko	Mago	*Melee*	0:18:05	2
12	Kindred	Atirador	Distância	0:18:31	1
13	Tristana	Atirador	Distância	0:14:43	0
14	Caitlyn	Atirador	Distância	0:19:53	0
15	Miss Fortune	Atirador	Distância	0:15:22	1
16	Tryndamere	Lutador	*Melee*	0:22:18	0
17	Annie	Mago	Distância	0:25:57	0
18	Aurelion Sol	Mago	Distância	0:25:22	0
19	Jarvan IV	Tanque	*Melee*	0:24:50	0
20	Mordekaiser	Lutador	*Melee*	0:25:08	1

A.2 Tabelas de Teste de Eficiência do Experimento 2

Tabela A.2. Execução de 10 partidas com a variável ϕ desativada. São listados os heróis utilizados, a duração da partida, a classe, a natureza do ataque, o número de Mortes sofridas, o número de abates realizados e a quantidade de tropas abatidas. A duração das partidas encontra-se no formato hh:mm:ss.

# Execução	Herói	Duração	Classe	Ataque	Mortes	Abates	Tropas
1	Caitlyn	0:25:30	Atirador	Distância	0	3	196
2	Annie	0:32:30	Mago	Distância	1	2	135
3	Jinx	0:24:07	Atirador	Distância	0	2	187
4	Twisted Fate	0:30:40	Mago	Distância	1	0	137
5	Vayne	0:25:21	Atirador	Distância	0	4	185
6	Tryndamere	0:37:27	Lutador	*Melee*	2	0	212
7	Jax	0:35:32	Lutador	*Melee*	1	0	182
8	Jarvan IV	0:35:58	Tanque	*Melee*	1	0	196
9	Fiora	0:32:04	Lutador	*Melee*	1	1	238
10	Wukong	0:39:27	Lutador	*Melee*	1	0	271

Tabela A.3. Execução de 10 partidas com a variável ϕ ativada. São listados os heróis utilizados, a duração da partida, a classe, a natureza do ataque, o número de Mortes sofridas, o número de abates realizados e a quantidade de tropas abatidas. A duração das partidas encontra-se no formato hh:mm:ss

# Execução	Herói	Duração	Classe	Ataque	Mortes	Abates	Tropas
1	Caitlyn	0:21:33	Atirador	Distância	0	5	210
2	Annie	0:31:37	Mago	Distância	0	0	251
3	Jinx	0:22:12	Atirador	Distância	0	2	221
4	Twisted Fate	0:29:27	Mago	Distância	1	1	267
5	Vayne	0:24:02	Atirador	Distância	0	3	241
6	Tryndamere	0:39:13	Lutador	*Melee*	3	0	362
7	Jax	0:32:31	Lutador	*Melee*	0	0	305
8	Jarvan IV	0:32:33	Tanque	*Melee*	1	0	296
9	Fiora	0:31:52	Lutador	*Melee*	1	1	287
10	Wukong	0:37:34	Lutador	*Melee*	0	0	351

Apêndice B

Sistema de Dicas ao Jogador

A tabela B.1 mostra as regras, temporizadores e dicas possíveis. Estas dicas foram utilizadas para a implementação do sistema de dicas ao jogador. Os intervalos entre as dicas encontram-se representados em segundos (s). As dicas encontram-se separadas, uma por linha, sendo uma destas selecionadas aleatoriamente após a ativação da regra. Erros tipográficos e abreviações foram inseridos aleatoriamente nas dicas, a fim de simular a digitação de mensagens.

Tabela B.1. Tabela de dicas ao jogador

Regra	Intervalo (s)	Possíveis Dicas
Escapar do Inimigo com W	600	"Usa o W para fugir!" "Foge com o W" "O W te ajuda a fugir" "Usa o W pra trás"
Alvo de herói inimigo	180	"Cuidado com os herois inimigos" "Foge deles!" "Cuidado que eles estão te tando dano."
Alvo de torre inimiga	180	"Cuidado com a torre!" "A torre está te atacando, cuidado!" "Sai da parte vermelha" "Foge da torre!"
Foco de tropas inimigas	180	"Os minions estão te atacando, melhor recuar" "Cuidado com os minions" "Os minios vão te matar"
Pouco HP	240	"Vc está com hp baixo, vai base!" "Melhor vc ir base" "Aperta B e vai base" "Vc ta com pouca vida, melhor ir base" "Se você tiver poção usa, senão melhor ir base"
Last hit	180	"Tenta acertar o último ataque nos minions." "Dá o último golpe nos minions que vc vai ganhar ouro"
Posicionamento avançado	180	"Joga mais recuado." "Tenta jogar mais perto da torre"
Alto uso de mana	600	"Voce está gastando muita mana" "Tenta economizar mana" "Guarda mana para usar poder nos inimigos"
Farming com Q	600	"Usa o Q para farmar" "Usa o Q nos minions pra matar" "Se vc usar o Q fica mais facil farmar"
Harass com Q	600	"Usa o Q para dar dano nos bots" "Tenta acertar o Q nos inimigos" "Dá Q nos inimigos"
Abate com R	600	"Usa R pra matar o heroi inimigo" "Mata ele com o R" "Dá R nele"
Estímulo por bom desempenho	300	"Nice!" "GJ" "Massa" "Bom trabalho" "Boa!"

Apêndice C

Questionário Aplicados a Jogadores Humanos

Este apêndice apresenta os questionários aplicados aos jogadores humanos durante os testes descritos na Seção 5.3.2. Após o convite para participar dos experimentos, os jogadores foram solicitados a preencher um questionário pré-teste que buscava traçar o perfil de jogador a ser testado. Após o preenchimento deste questionário, o jogador foi convidado a jogar duas partidas de League of Legends no mapa de *Summoner's Rift* na companhia do agente desenvolvido. Após finalizar as duas partidas, requereu-se do jogador o preenchimento de um questionário pós-teste. Estes questionários são apresentados a seguir.

C.1 Questionário Pré-Teste

O questionário pré-teste tem por objetivo traçar o perfil do usuário a ser testado. Ele compreende questões que cobrem dados pessoais do jogador, como visto na Figura C.1, a experiência do jogador com MOBAs em geral, como visto na Figura C.2, e a experiência do jogador com LoL, observado na Figura C.3. A exibição do questionário presente na Figura C.3 está condicionado à resposta "sim"à ultima questão do questionário presente na Figura C.2.

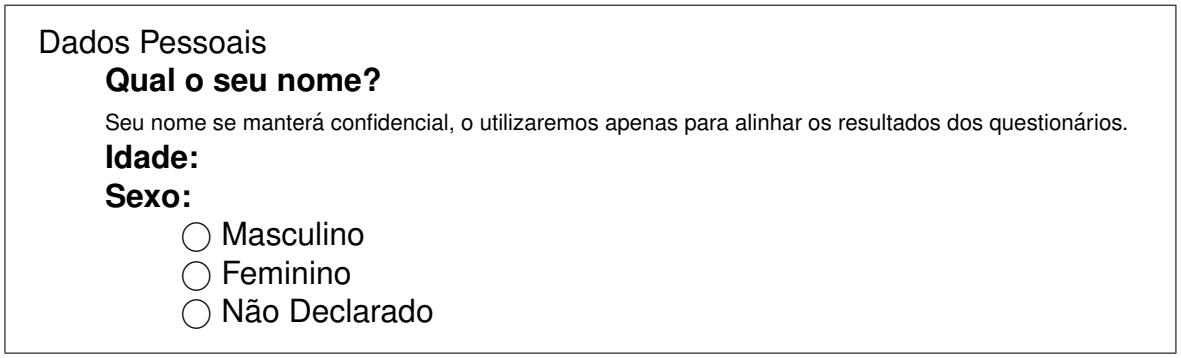

Figura C.1. Dados pessoais do jogador.

Questões relativas à sua experiência com MOBAs

Você já jogou algum MOBA?

e.g. Dota2, DotA, LoL, HoN, etc.

- ◯ Sim
- ◯ Não

Em relação à sua expertise em jogos MOBA:

- ◯ Sou iniciante
- ◯ Sou intermediário
- ◯ Sou avançado
- ◯ Sou profissional
- ◯ Não jogo MOBA

Há quanto tempo você joga MOBA?

- ◯ Nunca Joguei
- ◯ Menos de um mês
- ◯ Menos de seis meses
- ◯ Menos de um ano
- ◯ Menos de três anos
- ◯ Menos de cinco anos
- ◯ Menos de dez anos

Quais MOBAs você já jogou?

Marque todos que se aplicam.

- ☐ Defense of the Ancients
- ☐ Heroes of Newerth
- ☐ Dota2
- ☐ Strife
- ☐ League of Legends
- ☐ Heroes of the Storm
- ☐ Smite
- ☐ Outros

Você ja jogou ou joga League of Legends?

- ◯ Sim
- ◯ Não

Figura C.2. Questões relativas à experiência do jogador para com jogos MOBA.

Questões relativas à sua experiência com League of Legends.

Qual o seu nome de invocador em League of Legends?

Qual o seu nível em League of Legends?

Qual a sua posição favorita em League of Legends?

- ◯ Mid Laner
- ◯ Top Laner
- ◯ AD Carry
- ◯ Suporte
- ◯ Jungler

Você conhece o herói Soraka?

- ◯ Sim
- ◯ Não

Você ja jogou com o herói Soraka?

- ◯ Sim
- ◯ Não

Você fez o tutorial de League of Legends quando começou a jogar?

- ◯ Sim
- ◯ Não

Você acha o tutorial suficiente para ensinar um jogador iniciante sobre o jogo?

- ◯ Sim
- ◯ Não

Figura C.3. Questões relativas à experiência do jogador para com características de League of Legends.

C.2 Questionário Pós Teste

Após as partidas, os jogadores foram solicitados a completar um questionário pós-jogo. Neste questionário procurou-se mensurar a qualidade do sistema de suporte ao jogador e o sistema de dicas (Figura C.4). Além disso, o jogador foi questionado sobre o comportamento do agente durante a partida, validando sua perícia e nível de auxílio (Figura C.5).

Qual o seu nome?

Seu nome se manterá confidencial, o utilizaremos apenas para alinhar os resultados dos questionários.

Dê uma nota sobre o quão interessante, em relação à ajuda a novos jogadores, você imagina que o sistema apresentado seria:

0 para muito desinteressante, 10 para muito interessante.

0 ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ 10

Você achou o sistema de dicas importante?

◯ Sim
◯ Não

Que mecânicas de jogo você acha que o sistema poderia auxiliar?

Marque todas as aplicáveis.

☐ *Farming*
☐ *Harass*
☐ *Jungling*
☐ *Teamfight*
☐ *Awareness*

Em relação às dicas apresentadas, dê uma nota sobre o quão úteis você considera que elas seriam para novos jogadores:

0 para muito inúteis, 10 para muito úteis.

0 ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ ◯ 10

Em relação à frequência das dicas, você diria que:

◯ São muito frequentes, requerem menor frequência
◯ Requerem leve redução de frequência
◯ Têm frequência ideal
◯ Requerem leve aumento de frequência
◯ São pouco frequentes, requerem maior frequência

Figura C.4. Questões sobre o sistema de suporte e o sistema de dicas ao jogador.

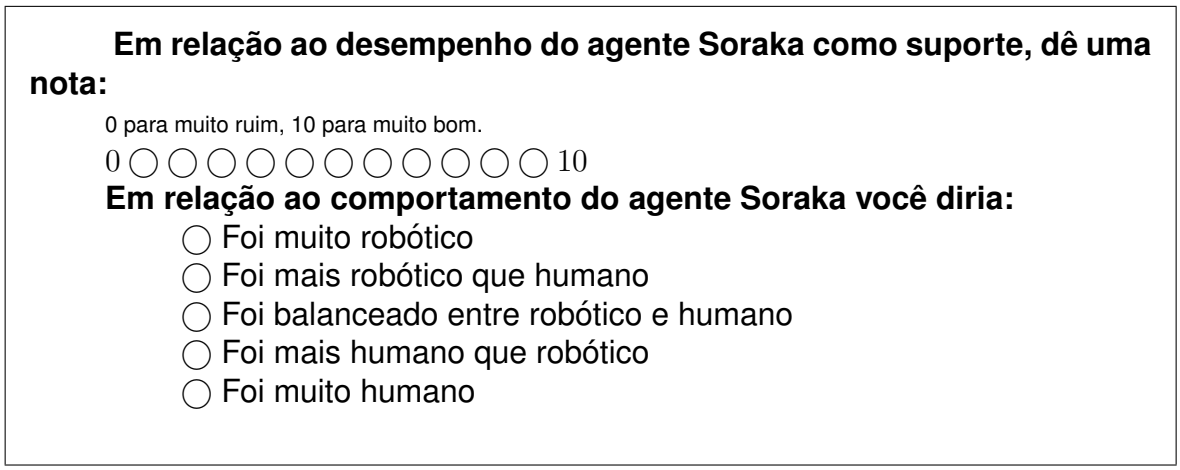

Figura C.5. Questões sobre o comportamento do herói durante a partida.

Referências Bibliográficas

Bevilacqua, F. (2013). http://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation–gamedev-11867.

Buro, M. & Churchill, D. (2012). Real-time strategy game competitions. *AI Magazine*, 33(3):106.

Champandard, A.; Dill, K. & Isla, D. (2011). Lay of the land: Smarter ai through influence maps. Em *Proceedings of Game Developers Conference*.

Cunha, R. L. D. F.; Machado, M. C. & Chaimowicz, L. (2015). Rtsmate: Towards an advice system for rts games. *Computers in Entertainment (CIE)*, 11(4):1.

Drachen, A.; Yancey, M.; Maguire, J.; Chu, D.; Wang, I. Y.; Mahlmann, T.; Schubert, M. & Klabajan, D. (2014). Skill-based differences in spatio-temporal team behaviour in defence of the ancients 2 (dota 2). Em *Proceedings of the IEEE Games, Entertainment, and Media (GEM) Conference 2014*.

Ferrari, S. (2013). From generative to conventional play: Moba and league of legends. Em *Proceedings of DeFragging Game Studies*, Atlanta, GA, USA. DiGRA.

Gaudiosi, J. (2012). Riot games' league of legends officially becomes most played pc game in the world. *Forbes. Jul*, 11:2011.

Hagelbäck, J. & Johansson, S. J. (2008a). The rise of potential fields in real time strategy bots. Em *Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference*, volume 8, pp. 42--47.

Hagelbäck, J. & Johansson, S. J. (2008b). Using multi-agent potential fields in real-time strategy games. Em *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 631--638.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88--105.

Ierusalimschy, R.; De Figueiredo, L. H. & Celes Filho, W. (1996). Lua - an extensible extension language. *Softw., Pract. Exper.*, 26(6):635--652.

Koster, R. (2013). *Theory of fun for game design*. "O'Reilly Media, Inc.".

Kwak, H.; Blackburn, J. & Han, S. (2015). Exploring cyberbullying and other toxic behavior in team competition online games. *Social Dynamics*, 22(28):47.

Laird, J. E. (2002). Research in human-level ai using computer games. *Communications of the ACM*, 45(1):32--35.

Liu, S.; Louis, S. J. & Ballinger, C. (2014). Evolving effective micro behaviors in RTS game. Em *Proceedings of Computational Intelligence and Games*, pp. 1–8.

Millington, I. & Funge, J. (2012). *Artificial intelligence for games*. "CRC Press".

Murphy, M. (2015). MOST PLAYED GAMES: NOVEMBER 2015 – FALLOUT 4 AND BLACK OPS III ARISE WHILE STARCRAFT II SHINES. http://caas.raptr.com/most-played-games-november-2015-fallout-4-and-black-ops-iii-arise-while-starcraft-ii-shines/Raptr.com[Online; posted 21-December-2015].

Nosrati, M. & Karimi, R. (2013). General trends in multiplayer online games. *World Applied Programming*, 3(1):1–4.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D. & Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. *Proceedings of Computational Intelligence and AI in Games*, 5(4):293--311.

Orkin, J. (2004). Applying goal-oriented action planning to games. *AI Game Programming Wisdom*, 2(2004):217--227.

Orkin, J. (2006). Three states and a plan: the ai of fear. Em *Proceedings of Game Developers Conference*, volume 2006, p. 4.

Pobiedina, N.; Neidhardt, J.; Moreno, M. d. C. C.; Grad-Gyenge, L. & Werthner, H. (2013). On successful team formation: Statistical analysis of a multiplayer online game. Em *Proceedings of Business Informatics*, pp. 55--62. IEEE.

Rioult, F.; Métivier, J.-P.; Helleu, B.; Scelles, N. & Durand, C. (2014). Mining tracks of competitive video games. Em *Proceedings of Conference on Sports Engineerings and Computer Science*, volume 8, pp. 82--87. Elsevier.

Silva, M.; Silva, V. N. & Chaimowicz, L. (2015). Dynamic difficulty adjustment through an adaptive AI. Em *Proceedings of Brazilian Symposium on Games and Entertainment*.

Silva, M. P.; Silva, V. N. & Chaimowicz, L. (2016). Dynamic difficulty adjustment on MOBA games. *Entertainment Computing (Submetido)*.

Silva, V. N. & Chaimowicz, L. (2015a). On the dvelopment of intelligent agents for MOBA games. Em *Proceedings of Brazilian Symposium on Games and Entertainment*.

Silva, V. N. & Chaimowicz, L. (2015b). A Tutor Agent for MOBA Games. Em *Proceedings of Brazilian Symposium on Games and Entertainment (SBGames)*.

Stanescu, M.; Hernandez, S. P.; Erickson, G.; Greiner, R. & Buro, M. (2013). Predicting army combat outcomes in starcraft. Em *Proceedings of AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Stanescu, M.; Hernandez, S. P.; Erickson, G.; Greiner, R. & Buro, M. (2014). Generating an attribute space for analyzing balance in single unit rts game combat. Em *Proceedings of Computational Intelligence and Games (CIG)*.

Tavares, A. R.; Azpúrua, H. & Chaimowicz, L. (2014). Evolving swarm intelligence for task allocation in a real time strategy game. Em *Proceedings of Brazilian Symposium on Computer Games and Digital Entertainment*.

Tozour, P. (2001). Influence mapping. *Game programming gems*, 2:287--297.

Uriarte, A. & Ontañón, S. (2012). Kiting in rts games using influence maps. Em *Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference*.

Weber, B. G.; Mateas, M. & Jhala, A. (2011). Building human-level ai for real-time strategy games. Em *Proceedings of AAAI Fall Symposium: Advances in Cognitive Systems*.

Willich, J. (2015). Reinforcement learning for heroes of newerth. Monografia, Technische Universitat Darmstadt.

Yang, P.; Harrison, B. & Roberts, D. L. (2014). Identifying patterns in combat that are predictive of success in moba games. Em *Proceedings of Foundations of Digital Games*.