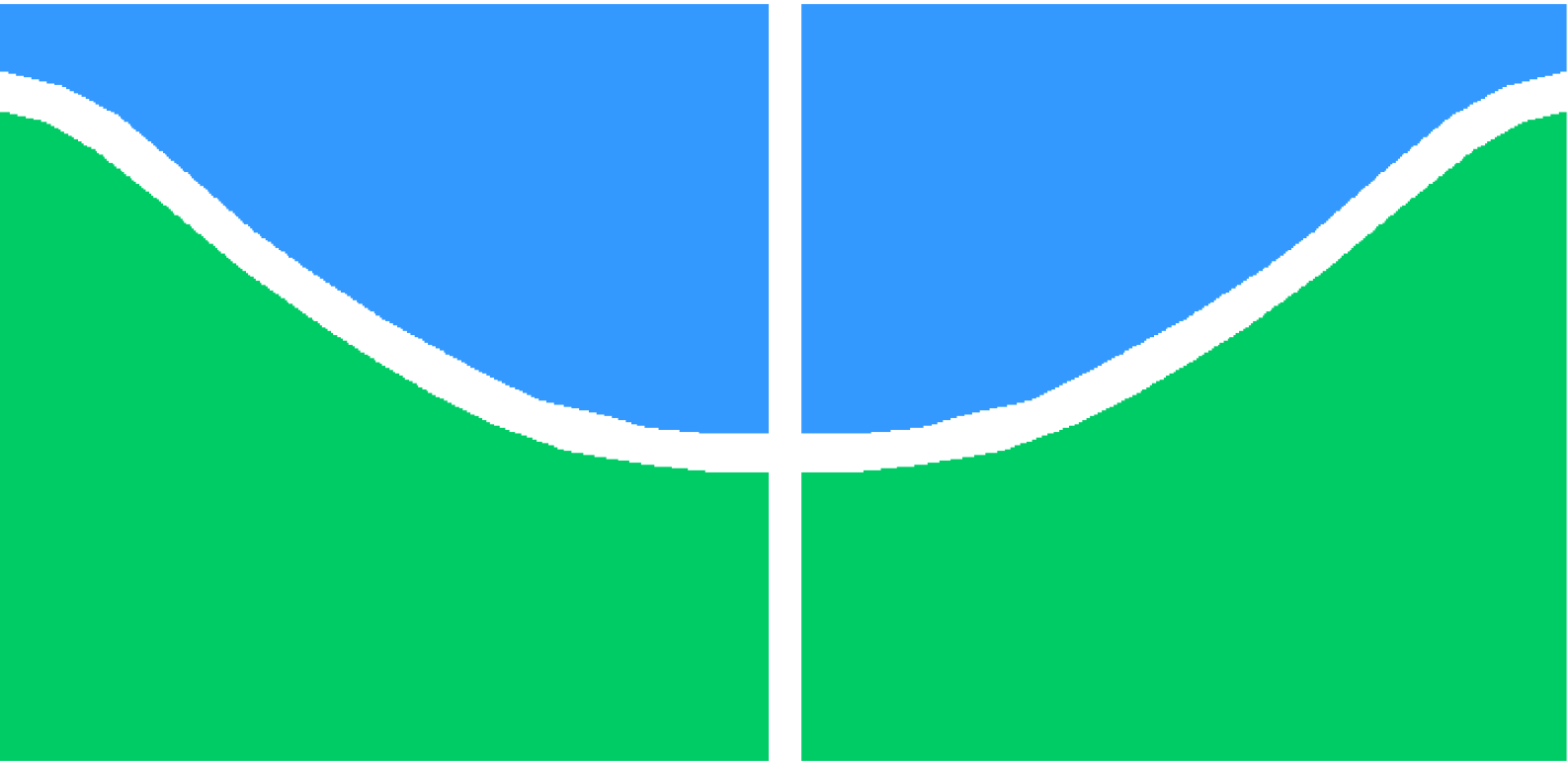

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Desenvolvimento de Jogadores Automatizados utilizando Máquina de Estados, Comportamentos de Direção e Algoritmos de Busca em Grafos

Autor: Bruno Rodrigues de Andrade

Orientador: Profª. Drª. Milene Serrano

Brasília, DF

2015

Bruno Rodrigues de Andrade

Desenvolvimento de Jogadores Automatizados utilizando Máquina de Estados, Comportamentos de Direção e Algoritmos de Busca em Grafos

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Profª. Drª. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2015

Bruno Rodrigues de Andrade

Desenvolvimento de Jogadores Automatizados utilizando Máquina de Estados, Comportamentos de Direção e Algoritmos de Busca em Grafos/ Bruno Rodrigues de Andrade. – Brasília, DF, 2015-

147 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª. Drª. Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Jogadores Automatizados. 2. Máquina de Estados. I. Profª. Drª. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de Jogadores Automatizados utilizando Máquina de Estados, Comportamentos de Direção e Algoritmos de Busca em Grafos

CDU 02:141:005.6

Bruno Rodrigues de Andrade

Desenvolvimento de Jogadores Automatizados utilizando Máquina de Estados, Comportamentos de Direção e Algoritmos de Busca em Grafos

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Trabalho aprovado. Brasília, DF, 02 de dezembro de 2015:

Profª. Drª. Milene Serrano
Orientador

Prof. Dr. Maurício Serrano
Coorientador

Prof. Dr. Edson Alves da Costa Júnior
Convidado 1

Prof. Dr. Fábio Macêdo Mendes
Convidado 2

Brasília, DF
2015

Agradecimentos

Agradeço a Deus acima de tudo, que me deu a oportunidade única de estudar em uma universidade federal e tem me sustentado em toda essa caminhada. Sem a ajuda Dele, todo o meu esforço teria sido em vão.

Agradeço a meus pais, Júlio Falcão e Denise Nery, que desde antes de eu ingressar na universidade sempre apoiaram meus sonhos e objetivos, proporcionando um suporte emocional indispensável em toda a minha carreira acadêmica. Sou muito grato pelo investimento deles e sua preocupação com a minha formação e vida profissional.

Agradeço a meu irmão Matheus Rodrigues que desde criança foi meu companheiro e amigo, fazendo parte da minha história desde o começo.

Agradeço à todo o corpo acadêmico da Falcudade UnB Gama por terem contribuído na minha formação acadêmica ao longo de cinco anos ampliando meus conhecimentos sobre Engenharia de Software e contribuindo para me tornar um profissional qualificado no mercado de trabalho.

Agradeço à professora Milene Serrano e ao professor Maurício Serrano que com excelência e dedicação me orientaram e guiaram por todo esse processo de escrita deste Trabalho de Conclusão de Curso, sempre me motivando à realizar um excelente trabalho desde o começo.

Agradeço os meus colegas e amigos de trabalho Felipe Perius, Henrique Prudêncio, Luís Rezende e Thamara Gabriella que trabalharam comigo no desenvolvimento do jogo *Lost Gems*, tema deste Trabalho de Conclusão de Curso.

Agradeço à Joice Rute Alves que também sempre esteve perto me animando e motivando para conquistar meus objetivos a fim de que possamos construir uma longa vida juntos no futuro.

Agradeço aos demais amigos e companheiros que também proveram um suporte emocional e intelectual importantíssimo para a minha formação acadêmica e profissional.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito."
(Bíblia Sagrada, Romanos 12, 2)

Resumo

O estudo de jogadores autônomos tem sido amplamente discutido desde a aparição de jogos eletrônicos ao redor do mundo. Vários cientistas e engenheiros têm empreendido esforços para definir uma Inteligência Artificial de jogadores automatizados adequada em contextos e jogos diferentes. O presente trabalho descreve uma proposta de desenvolver jogadores automatizados para o jogo *Lost Gems* (desenvolvido por Bruno Rodrigues, Luís Rezende, Felipe Perius, Henrique Prudêncio e Thamara Gabriella para a plataforma iOS). Estes jogadores possuem uma série de comportamentos pré-programados: percorrer o mapa, atirar em inimigos, capturar um objeto na base inimiga, entre outros. Essas estratégias foram implementadas utilizando-se de Máquina de Estados Finitos, Comportamentos de Direção, Grafos, Algoritmo de Busca de Menor Caminho. Os jogadores automatizados devem agir simulando o comportamento de jogadores humanos a fim de que o jogo se torne mais competitivo, e equilibrado na sua dificuldade.

Palavras-chaves: inteligência artificial. jogadores automatizados. algoritmos. máquina de estados finitos. comportamentos de direção. grafos.

Abstract

The study of autonomous agents has been widely discussed the emergence of electronic games around the world. Several scientists and scholars have made efforts to set a properly Artificial Inteligence of autonomous players in different contexts and games. This paper describes a proposal to develop automated agents for the game Lost Gems (developed by Bruno Rodrigues, Luís Rezende, Felipe Perius, Henrique Prudêncio and Thamara Gabriella for the iOS plataform). These agents have a number of pre-programmed behaviors: scroll the map, shoot in enemies, capture an object in the enemy base, among others. These strategies was implemented using Finite State Machine, Steering Behaviours, graphs, Shortest Path Algorithm. Automated players should act simulating the behavior of human players in order that the game becomes more competitive and balanced in its difficulty.

Key-words: artificial inteligence. automated players. algorithms. finit state machines. steering behaviours. graphs.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA	Inteligência Artificial
NPC	*Non-Player Character*
FSM	*Finit State Machine*
TCC	Trabalho de Conclusão de Curso
GoF	*Gang of Four*
IDE	*Integrated Development Environment*
BSD	*Berkeley Software Distribution*
XNU	*X is Not Unix*
GUI	*Graphical User Interface*

Lista de símbolos

$\in$	Pertence
L	Lista de Nós
G	Grafo
V	Vértice
A	Aresta
Fd	Vetor de Força Direcional
Vd	Vetor de Velocidade Desejada
Va	Vetor de Velocidade Atual
P	Vetor Posição

Sumário

1 Introdução

Este capítulo tem como objetivo introduzir o leitor no assunto deste trabalho. São discutidos o Contexto do Trabalho, a Questão de Pesquisa, a Justificativa, o Objetivo Geral, os Objetivos Específicos, e a Organização do Documento.

1.1 Contextualização

Lost Gems[1] é um jogo eletrônico desenvolvido para a plataforma iOS por cinco desenvolvedores: Bruno Rodrigues, Luís Rezende, Felipe Perius, Henrique Prudêncio e Thamara Gabriella. O desenvolvimento se deu entre novembro de 2014 e março de 2015 durante o projeto BEPiD (*Brazillian Educational Program for iOS Developers*)[2]. A Figura 1 mostra uma imagem retirada diretamente de *Lost Gems*. A seguir, será apresentado, em linhas gerais, o contexto do jogo.

Figura 1: Imagem do Jogo

Basicamente, o jogo prevê duas naves espaciais, as quais levam diversos personagens. Essas naves caem, acidentalmente, em um planeta considerado hostil. As naves necessitam de duas gemas mágicas (ou seja, são artefatos que funcionam como combustível para a nave) para escaparem do planeta. Entretanto, quando as naves caíram, cada qual teve uma gema mágica destruída no acidente. Agora, para os personagens e as naves saírem do planeta, os mesmos devem se enfrentar, no intuito de conquistar a gema mágica da equipe inimiga, trazendo-a de volta à nave (estilo *capture the flag*). Nesse caso, ape-

[1] PERIUS, Felipe. LostGemsBigFinal. Brasília: 2015. 13 slides, color
[2] <http://www.bepid.org.br>

nas uma equipe sairá vencedora dessa competição. Ao longo da partida, os personagens podem ferir os inimigos com tiros bem como com habilidades específicas.

Vários jogadores podem se conectar no jogo via rede *Wi-Fi* ou *Bluetooth*, formando assim equipes diferentes, com quantidades de jogadores variáveis. Com isso, o jogo visa estabelecer uma interação maior entre os jogadores aumentando a competitividade e a diversão.

Um problema evidente que pode ocorrer é a questão do balanceamento do jogo quando, por exemplo, um time possui mais jogadores que outro. Nesse caso, uma equipe obterá vantagem numérica sobre a outra.

Outro problema evidente que acontece no jogo é que um jogador não pode jogar Lost Gems sozinho (um modo *Single Player*), pois é necessário que outros jogadores se conectem à partida para que esta ocorra.

1.2 Motivação

Com base nos problemas apresentados no tópico anterior, pode-se levantar a seguinte questão de pesquisa para motivação: É adequado desenvolver jogadores automatizados no Lost Gems, utilizando-se de Máquina de Estados, Comportamentos de Direção, Grafos e Algoritmos de Busca, visando simular o comportamento de jogadores humanos e equilibrar as equipes envolvidas em uma partida, mantendo a competitividade do jogo?

1.3 Justificativa

Para este trabaho, alguns algoritmos que dão suporte à Inteligência Artificial de jogadores automatizados serão utilizados.

Muitos jogos têm agentes controlados por computador que jogam contra um jogador. O comportamento desses agentes é descrito por meio da Inteligência Artificial (IA) no jogo. A IA é um componente importante do jogo, e precisa ser desenvolvida com cuidado e adaptada com regularidade (HASTJARJANTO; JEURING; LEATHER, 2013).

A Inteligência Artificial de jogos passou por uma revolução silenciosa nos últimos anos. Já não é mais algo que a maioria dos desenvolvedores consideram apenas para o fim do projeto, quando o prazo de conclusão está acabando e o publicador está pressionando para que o jogo seja disponibilizado no mercado. Mais recentemente, a Inteligência Artificial de jogos é algo que é planejado, algo que os desenvolvedores estão deliberadamente a tornando tão importante quanto os gráficos ou efeitos sonoros. Adicionalmente, vale salientar que o mercado está repleto de jogos de todos os tipos, e os desenvolvedores estão procurando refinar seus jogos no intuito de torná-los mais reconhecidos. Um

jogo com oponentes verdadeiramente "espertos" ou personagens que não são jogadores é automaticamente notado, não importando, muitas vezes, como ele é ou como se parece (BUCKLAND, 2005).

1.4 Objetivo Geral

Desenvolver jogadores automatizados no Lost Gems, utilizando-se de Máquina de Estados, Comportamentos de Direção, Grafos e Algoritmos de Busca, visando simular o comportamento de jogadores humanos e equilibrar as equipes envolvidas em uma partida, mantendo a competitividade do jogo.

1.5 Objetivos Específicos

No intuito de atingir o objetivo geral, alguns objetivos específicos merecem menção, dentre eles:

- Modelar e desenvolver uma máquina de estados que permita trabalhar a Inteligência Artificial dos jogadores automatizados;
- Estudar e aplicar algoritmos de busca em grafos, os quais podem contribuir com as estratégias implementadas nos jogadores automatizados;
- Implementar estratégias específicas para viabilizar algumas ações por parte dos jogadores automatizados, como se os mesmos estivessem sendo controlados por jogadores humanos. Nesse cenário, destacam-se os seguintes comportamentos:
  - Andar pelo mapa;
  - Desviar de obstáculos;
  - Capturar a gema inimiga na base inimiga;
  - Levar a gema capturada na base inimiga à base aliada;
  - Enfrentar jogadores da equipe inimiga;
  - Recuperar gema de sua equipe, a qual se encontra fora da base;
  - Defender a gema de sua equipe, para que a mesma não seja capturada pela equipe inimiga, e
  - Possuir dificuldade customizável pelo jogador.
- Coletar as primeiras impressões, junto aos usuários, visando investigar se os jogadores automatizados se comportam como jogadores humanos ou mesmo equilibram as equipes, e

- Estabelecer uma forma organizada, estruturada e orientada às boas práticas da Engenharia de Software no intuito de conduzir o processo investigativo, a implementação e a coleta das primeiras impressões.

1.6 Organização do Documento

Este documento está organizado em capítulos com sub-tópicos. São estes:

- **Referencial Teórico**: Neste capítulo, encontra-se uma fundamentação teórica dos conteúdos que são utilizados ao longo da realização desse TCC.
- **Suporte Tecnológico**: São descritas quais ferramentas foram utilizadas para o desenvolvimento do trabalho como um todo.
- **Metodologia**: Neste capítulo, são descritos os passos metodológicos para o desenvolvimento do trabalho.
- **Prova de Conceito**: É descrito como as estratégias de implementação foram escolhidas. Também é descrita uma prova de conceito.
- **Desenvolvimento dos Jogadores Automatizados**: Os algoritmos que suportam a Inteligência Artificial dos jogadores automatizados são descritos, relacionando-os com as classes implementadas.
- **Resultados Obtidos**: Nesse capítulo, são descritos os testes unitários (com cobertura de código), a análise do código fonte, e os testes com os usuários. Adicionalmente, são apresentados os resultados obtidos, retomando a questão de pesquisa.

2 Referencial Teórico

Neste capítulo, será apresentada uma descrição detalhada dos conceitos e conteúdos que foram trabalhados ao longo do desenvolvimento deste TCC. Todos os conceitos estão embasados em artigos científicos e livros publicados na área de Tecnologia da Informação.

O capítulo está organizado em seções com focos específicos, no caso: referencial teórico orientado à Inteligência Artificial; ao domínio de jogos e orientado à Engenharia de Software.

2.1 Inteligência Artificial

Inteligência Artificial (IA) é uma ciência tecnológica, que pesquisa e desenvolve métodos, técnicas e aplicações para simular, extender e expandir a teoria da inteligência humana. IA é um ramo da ciência da computação que tenta entender a essência da inteligência e produzir uma nova máquina inteligente que pode realizar reações similares à inteligência humana (NING; YAN, 2010).

De acordo com o autor Li (2009), a IA é: um sistema computacional que possui conhecimento e comportamento humano com habilidades como por exemplo: aprender, inferir, julgar, resolver o problema, memória, conhecimento e entender a linguagem natural humana.

Com base no trabalho dos autores Hosea, Harikrishnan e Rajkumar (2011), a Inteligência Artificial abrange:

- **Jogos eletrônicos**: programar computadores para jogar jogos como xadrez e damas, os quais demandam algoritmos baseados em raciocínio lógico;
- **Sistemas especialistas**: programar computadores para tomar decisões em situações da vida real (por exemplo, alguns sistemas especialistas ajudam médicos a diagnosticarem doenças baseados em sintomas);
- **Linguagem natural**: programar computadores para entender a linguagem natural humana;
- **Redes neurais**: sistemas que simulam inteligência pela tentativa de reproduzir os tipos de conexões físicas que ocorrem em cérebros de animais, e
- **Robótica**: programar computadores para perceber e reagir a outros estimulos sensoriais.

2.1.1 Máquinas de Estados Finitos

Em jogos eletrônicos, Máquinas de Estados Finitos, ou do inglês *Finite State Machine* (FSM), são tipicamente usadas para modelar o comportamento de NPCs (*Non-Player-Character*), para fazê-los reagir a eventos do jogo o mais natural e inteligente possível. FSM consiste em um conjunto de estados, que representam algum tipo de ação ou comportamento, e uma coleção de transições para cada estado, que especificam as reações dos NPCs aos eventos do jogo (HU; ZHANG; MAO, 2011).

De fato, Máquinas de Estados Finitos são a abordagem mais popular para algoritmos de Inteligência Artificial em jogos, por várias razões, incluindo: facilidade de entendimento, facilidade de implementação, eficiência, entre outras. Facilidade de implementação também permite uma prototipação rápida e mais iterações de desenvolvimento. FSMs possuem a característica de serem leves e rápidas, o que é sempre importante em sistemas *real-time*. Comparado com outros esquemas de controle, FSMs são mais fáceis de testar e validar (HU; ZHANG; MAO, 2011).

Existem outros esquemas de controle equivalentes, como por exemplo a máquina de Moore e a de Mealy[1]. Essas máquinas não serão detalhadas nesse trabalho, por não serem foco do projeto.

2.2 Domínio de Jogos

2.2.1 Jogadores Automatizados

Neste trabalho, a expressão "Jogadores Automatizados" é considerada sinônima às expressões "Personagens Automatizados", "jogadors Automatizados" e *Non-player Characters* (NPCs).

Em geral, quando se fala em desenvolver comportamentos de um *bot*, o termo NPC refere-se a um jogador que é capaz de jogar um jogo por si só. No entanto, alguns autores restringem ainda mais a definição acima de um NPC, sendo esse considerado um personagem do jogo, o qual enfrenta um jogador humano. Já um jogador que joga por si só, sem oponentes humanos, é chamado simplesmente de *bot* (RECIO et al., 2012). Ao longo desse trabalho, os dois termos, NPCs e *bots*, serão utilizados como sinônimos.

Uma das muitas contribuições históricas da Inteligência Artificial aos jogos eletrônicos consiste em gerar jogadores autônomos ou NPCs. Em geral, um jogador se concentra em desenvolver estratégias em um jogo competitivo, ajustando o nível do jogo de acordo com as habilidades de um jogador humano, ou através do desenvolvimento de estratégias e comportamentos sememelhantes a humanos. *Bots* de um jogo podem ser criados por

[1] <http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Seq/fsm.html>

diversas técnicas de Inteligência Artificial (RECIO et al., 2012). Algumas dessas técnicas são citadas neste trabalho.

2.2.2 Comportamentos de Direção

No final de 1980, o cientista da computação Craig Reynolds desenvolveu o algoritmo *Steering Behaviours* (Comportamentos de Direção) para personagens animados. Esses comportamentos permitiram aos elementos individuais percorrerem em seus ambientes digitais de maneira "realista", com estratégias de fuga, vagueação, chegada, perseguição, escape, entre outras. Utilizado no caso de um único jogador autônomo, esses comportamentos são simples de serem entendidos e implementados (SHIFFMAN et al., 2012).

O comportamento de um jogador automatizado pode ser melhor entendido sendo dividido em várias camadas. Essas camadas são destinadas para clareza e especificidade na discussão que vai se seguir. A Figura 2 mostra uma divisão do comportamento de movimento para jogadores autônomos em uma hierarquia de três camadas: seleção de ação, direção e locomoção (REYNOLDS, 1999).

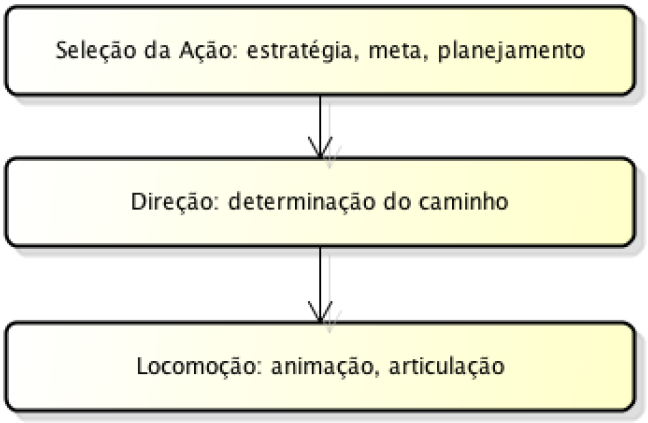

Figura 2: Uma Hierarquia de Comportamentos de Movimento.

Na camada Seleção de Ação, um jogador possui uma meta (ou metas) e pode selecionar uma ação (ou uma combinação de ações) baseada(s) naquela meta (SHIFFMAN et al., 2012). Reynolds (1999) descreve várias metas e ações associadas, tais como: buscar um alvo, evitar um obstáculo, e seguir um caminho.

Neste trabalho, várias dessas ações serão implementadas e combinadas com outros algoritmos. Outros detalhes serão apresentados no capítulo 4, Proposta.

Uma vez que uma ação foi selecionada, o jogador deve calcular seu próximo movimento (essa é a camada de Direção). Esse próximo movimento será influenciado por

uma força externa ao personagem; mais especificamente, a representação de uma força newtoniana (no âmbito computacional) que o direciona para uma posição qualquer dependendo do comportamento direcional selecionado (SHIFFMAN et al., 2012). Reynolds (1999) desenvolveu uma fórmula simples de força direcional que será utilizada ao longo deste trabalho, onde $\boldsymbol{F_d}$ é um vetor que representa a força que direciona o jogador, $\boldsymbol{V_d}$ é um vetor que representa a velocidade desejada (após ser calculada dependendo do comportamento do jogador automatizado), e $\boldsymbol{V_a}$ é um vetor que reflete a velocidade atual do jogador.

$$\boldsymbol{F_d} = \boldsymbol{V_d} - \boldsymbol{V_a} \tag{2.1}$$

A locomoção de um jogador autônomo pode ser baseada na representação de sua animação. Um personagem pode ser representado por uma simulação balanceada baseada na física de caminhar, proporcionando uma animação realista e uma locomoção comportamental. Ou um personagem pode possuir um modelo de locomoção muito simples para que uma representação estática (como uma nave espacial) ou pré-animada (como uma figura humana realizando um ciclo de caminhada) seja anexada. Resumindo, a locomoção pode ser restringida ao movimento inerente a um conjunto fixo de segmentos pré-animados (andar, correr, parar, virar à esquerda, entre outros) que são selecionados de forma discreta ou misturados (REYNOLDS, 1999).

A implementação de todas as forças envolvidas nos comportamentos de direção podem ser atingidos através de vetores matemáticos. A Figura 3 representa um personagem posicionado na coordenada *(x, y)* com uma velocidade *(a, b)* (BEVILACQUA, 2012).

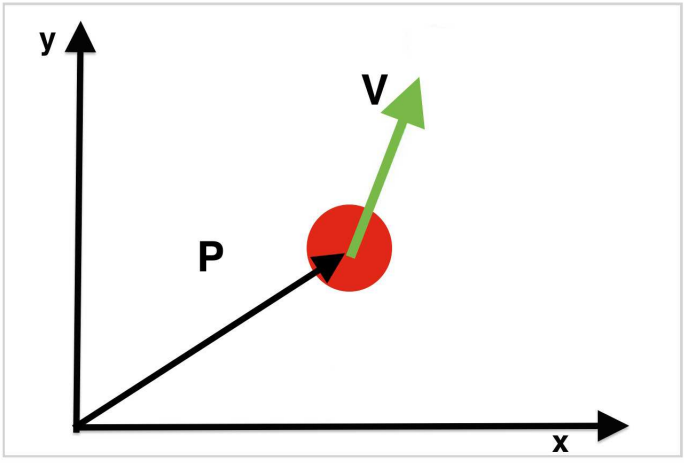

Figura 3: Vetores de Posição e Velocidade Sob Um Personagem.

O movimento é calculado usando a seguinte equação:

$$\boldsymbol{P_{i+1}} = \boldsymbol{P_i} + dt\boldsymbol{V_a} \tag{2.2}$$

Onde $\boldsymbol{P_i}$ é a posição atual, e *dt* é o tempo decorrido entre o *frame* anterior do jogo e o atual.

A direção do vetor velocidade irá controlar para onde o personagem está indo; enquanto seu comprimento (ou magnitude) irá controlar o quanto o personagem irá se mover a cada *frame*. Quanto maior for o comprimento, mais rápido o jogador se moverá. O vetor velocidade pode ser truncado para garantir que ele não será maior do que um determinado valor, geralmente, velocidade máxima (BEVILACQUA, 2012).

Utilizando dessas premissas matemáticas e físicas, vários Comportamentos de Direção podem ser derivados. Alguns destes serão utilizados na construção na Inteligência Artificial dos jogadors automatizados em *Lost Gems*. A seguir, são apresentadas descrições de alguns desses comportamentos.

2.2.2.1 Seguir

O comportamento Seguir (ou *Seek*) atua para orientar o personagem para uma posição específica no espaço global. Este comportamento ajusta o personagem de modo que sua velocidade está alinhada radialmente no sentido do alvo. Isso é diferente de uma força atrativa (como a gravidade) que produziria um caminho orbital ao redor do ponto de destino. A "velocidade desejada" é um vetor na direção do personagem para o alvo. O comprimento da "velocidade desejada" poderia ser a velocidade máxima definida para o *bot*, ou pode ser a velocidade atual, dependendo da aplicação específica. O vetor de direção (ou *steering vector*) é a diferença entre esta velocidade desejada e a velocidade atual do personagem (REYNOLDS, 1999).

As fórmulas de cálculo da força de direção (*steering*) (onde $\boldsymbol{P_a}$ é o vetor posição do alvo, V_{max} é a velocidade máxima definida para o jogador, e $\hat{\boldsymbol{P}}$ é o vetor unitário da diferença da posição do jogador e da posição do alvo) são:

$$\boldsymbol{P} = \boldsymbol{P_i} - \boldsymbol{P_a} \tag{2.3}$$

$$\boldsymbol{V_d} = V_{max}\hat{\boldsymbol{P}} \tag{2.4}$$

$$\boldsymbol{F_d} = \boldsymbol{V_d} + \boldsymbol{V_a} \tag{2.5}$$

Todos esses valores são calculados a cada quadro do jogo no seu *loop* principal. A força $\boldsymbol{F_d}$ atua diretamente no jogador automatizado, alterando a sua aceleração, e consequentemente, o seu vetor velocidade.

A Figura 4 ilustra o cálculo de vetores para o comportamento *Seguir*. O vetor pontilhado mostra como a adição da força direcional à velocidade atual produz o resultado desejado (BUCKLAND, 2005).

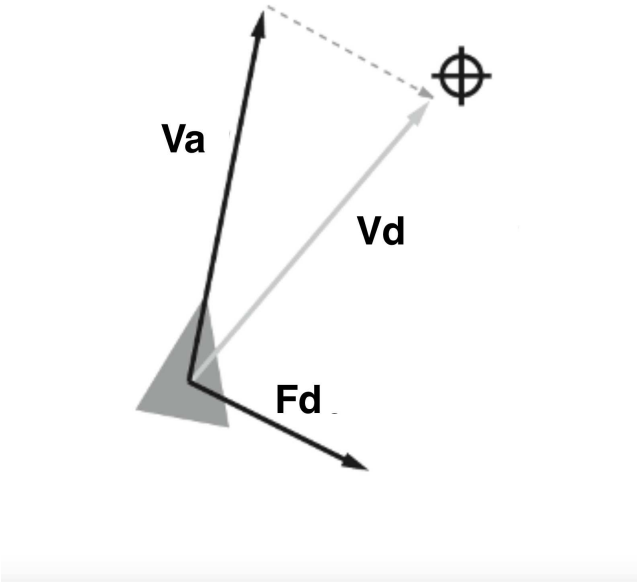

Figura 4: Representação das Forças Atuantes do Comportamento *Seguir* (BUCKLAND, 2005)

2.2.2.2 Fuga

Fuga (ou *Flee*) é simplesmente o inverso de *Seguir* e age para digirir o personagem de modo que a sua velocidade esteja alinhada radialmente para longe do alvo. Os pontos da velocidade desejada ficam na direção oposta (REYNOLDS, 1999).

A fórmula de cálculo da força de direção (*steering*) é:

$$\boldsymbol{F_d} = \boldsymbol{V_d} - \boldsymbol{V_a} \tag{2.6}$$

A Figura 5 ilustra o novo vetor $\boldsymbol{F_d}$ que é calculado subtraindo a posição do alvo, o que produz um vetor que vai do alvo para o personagem. (BEVILACQUA, 2012)

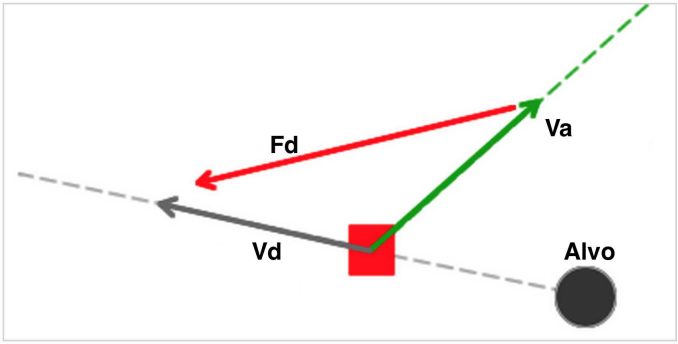

Figura 5: *Representação das Forças Atuantes do Comportamento Fuga*

2.2.2.3 Evitar Obstáculos

Evitar Obstáculos (ou Obstacle Avoidance) é um comportamento que direciona um jogador a evitar obstáculos que estejam em seu caminho. Um objeto é qualquer objeto que pode ser aproximado por um círculo. Isso pode ser alcançado direcionando um jogador de forma a manter a área de um retângulo - uma caixa de detecção, estendendo a frente do jogador - livre de colisões. A largura da caixa de detecção é igual ao raio delimitador do jogador, e seu comprimento é proporcional à velocidade atual do jogador - quanto mais rápido for, maior será a caixa de detecção (BUCKLAND, 2005).

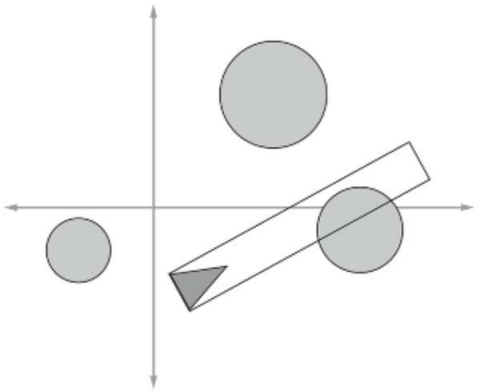

Figura 6: Representação do Comportamento *Evitar Obstáculos*

Para este trabalho, algumas adaptações foram feitas nesse comportamento. Pensou-se em um quadrado como uma caixa de detecção invisível que fica à frente do jogador automatizado. A sua posição é equivalente ao vetor velocidade do *bot*, conforme pode ser observado na Figura 7. Assim que a caixa de detecção colide com algum obstáculo do mapa, uma força direcional atua perpendicularmente para a direita do jogador, arrastando-o para o lado do obstáculo.

Quanto maior a velocidade do *bot*, mais longe a caixa estará do mesmo.

Esse algoritmo é executado constantemente no jogo, independentemente do estado dos *bots*.

Figura 7: Caixa de Detecção à Frente do *bot*

2.2.2.4 Seguir Caminho

Seguir Caminho (ou *Path Following*) cria uma força de direção que move um jogador por meio de uma série de pontos formando um caminho. Alguns caminhos possuem um ponto inicial e final, e outras vezes eles dão a volta em torno de si formando um caminho fechado e interminável (BUCKLAND, 2005).

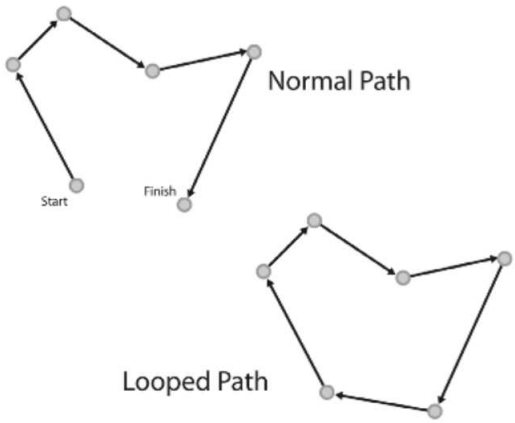

Figura 8: Diferentes Tipos de Caminho. (BUCKLAND, 2005)

Segundo BUCKLAND (2005), o modo mais simples de seguir um caminho é definir o ponto de passagem atual para o primeiro lugar em uma lista, usar o algoritmo *Seguir* até que o jogador alcance o ponto. Em seguida, deve-se definir o próximo ponto de passagem e usar o *Seguir* para alcançar o ponto e assim por diante, até que o ponto de passagem atual seja o último ponto na lista. Quando isso acontece, o jogador deve usar o algoritmo *Arrive* no ponto de passagem atual, ou, se o caminho é uma volta fechada, o ponto de passagem deve ser definido como o primeiro na lista novamente, e o jogador continua sucessivamente utilizando o algoritmo *Seguir*.

2.2.3 Grafos

2.2.3.1 Definição de Grafos

Um grafo simples é um par ordenado $G = (V, E)$, onde V é um conjunto finito cujos elementos são denomidados por vértices, e E é um conjunto de elementos do tipo x,y, com $x, y \in V$ e $x \neq y$, chamados arestas (LOPES, 2009).

Uma aresta como $\{v, w\}$ será denotada simplesmente vw ou por wv. Diremos que a aresta vw incide em v e em w e que v e w são vértices da aresta. Se vw é uma aresta, diremos que os vértices v e w são vizinhos ou adjacentes. De acordo com esta definição, um grafo não pode ter duas arestas diferentes com o mesmo par de pontas (ou seja, não pode ter arestas "paralelas"). Também não pode ter uma aresta com pontas coincidentes (ou seja, não pode ter "laços"). A Figura 9 ilustra um desenho de um grafo cujos vértices são t, u, v , w, x, y , z e cujas as arestas são vw, uv , xw, xu, yz e xy (FEOFILOFF et al., 2011).

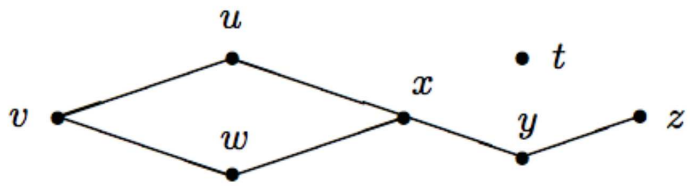

Figura 9: Representação de Um Grafo. (FEOFILOFF et al., 2011)

2.2.3.2 Lista de Adjacência

Um grafo $G = (V, A)$ pode ser representado por uma coleção de listas de adjacência ou por uma matriz de listas de adjacência. A lista de adjacência (que foi utilizada para se implementar o algorimo de travessia desse trabalho) consiste em um arranjo de adjacência de $||V||$ listas, uma para cada vértice em V. Para cada $u \in V$, a lista de adjacências Adj. [u] contém (índices para) todos os vértices v tais que existe uma aresta $(u, v) \in A$ (CORMEM et al., 2002). A Figura 10 a) representa um grafo não orientado que possui oito vértices e nove arestas. A Figura b) representa um grafo não orientado como uma lista de adjacências.

2.2.4 Busca em Largura

A busca em largura é um algoritmo, que pode ser usado como busca de vértices em um grafo, que utiliza uma fila para determinar a ordem de visitação dos vértices (*First in First Out*, isto é, o primeiro que entra é o primeiro que sai). Esta busca examina todos os vértices de certo nível do grafo antes dos vértices do nível abaixo. Se o grafo é finito e

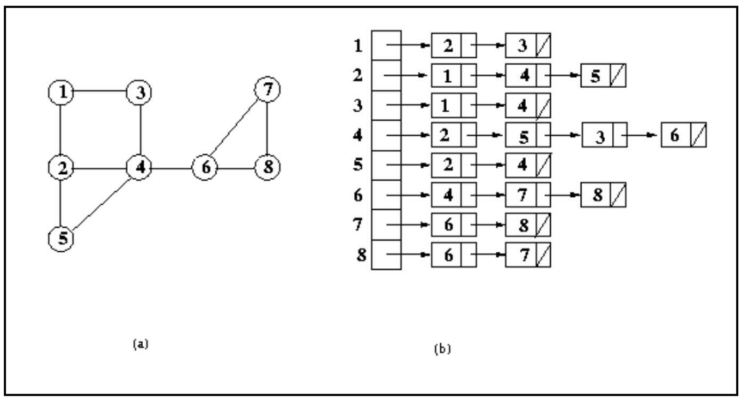

Figura 10: Representação de uma Lista de Adjacência. (CORMEM et al., 2002)

se existe uma solução, isto é, o vértice alvo pode ser alcançado, então ela será encontrada por este método (SILVA, 2005).

Em uma busca em largura a partir de um vértice v, espera-se que todos os vizinhos de v sejam visitados antes de continuar a busca mais profundamente. O algoritmo mostrado a seguir ilustra de forma suscinta este tipo de busca (SILVA, 2005).

```
1 Procedimento Busca(g: Grafo)
2         Para Cada vertice v de g:
3                 Marque um como nao visitado
4         Para Cada vertice v de g:
5                 Se v nao foi visitado:
6                         Busca_Profundidade(v)
7
8 Procedimento Busca_Profundidade(v: vertice)
9         Marque v como visitado
10        Para Cada vertice w adjacente a v:
11                Se w nao foi visitado:
12                        Busca_Profundidade(w)
```

2.2.5 Algoritmo A*

Segundo Rabin (2002), o algoritmo A* deve encontrar um caminho entre dois vértices em um grafo. Existem muitos algoritmos de busca, mas o A* encontra o menor caminho, se existir, e fará isto relativamente rápido – é isto que o diferencia dos outros. O algoritmo A* atravessa o grafo marcando vértices que correspondem às várias posições exploradas. Esses vértices gravam o progresso da busca. Além de guardar a posição do vértice em um espaço 2D, cada um desses vértices têm três atributos principais chamados comumente de F, G e H (SILVA, 2005):

- O atributo G de um determinado vértice é o custo que se tem desde o vértice de inicio até este vértice destino;
- O atributo H de um vértice é o custo estimado do caminho deste vértice até o vértice final. H entende-se por heurística e significa "estimativa bem comportada", visto que realmente não se conhece esse custo;
- O atributo F é a soma de G e H. F representa a melhor avaliação para o custo de um caminho passando através de um determinado vértice. Quanto menor o valor de F, para um determinado vértice, maiores são as chances de ele fazer parte do menor caminho.

O objetivo de F, G e H é quantificar o quão promissor é o caminho passando por determinado vértice. O atributo G pode ser calculado. Ele é o custo para chegar ao vértice corrente. Desde que se tenha explorado todos os vértices que apontam para este, sabe-se o valor exato de G. Entretanto, o atributo H é completamente diferente. Desde que não se saiba quanto além está o vértice final depois deste, deve-se estimar este valor. Na melhor das avaliações, o F fica mais próximo do valor correto, e mais rápido o A* encontra o fim com um pequeno esforço de empenho (RABIN, 2002).

O algoritmo A* mantém duas listas de vértices, uma para vértices abertos e outra para fechados. O vértice raiz é o primeiro a entrar na lista de abertos, e seu custo é marcado como zero. Como ele é o melhor e o único vértice da lista, ele é o primeiro a ser analisado.

2.3 Qualidade de Código

2.3.1 Complexidade Ciclomática

De acordo com a teoria dos grafos, o número ciclomático de um grafo G pode ser definido como (e - n + p), onde e é o número de arestas, n é o número de nós e p denota o número de componentes fortemente conectados. Na teoria dos grafos, a complexidade ciclomática representa o número de ciclos fundamentais de um grafo. No contexto de programas de computador, a complexidade ciclomática reflete o número de caminhos independentes em um programa. Mais recentemente, identifica as lógicas independentes utilizadas no programa e também provê uma medida quantitativa da complexidade lógica do software (TIWARI; KUMAR, 2014).

2.3.2 Complexidade Ciclomática de McCabe

Em 1976, McCabe (1976) desenvolveu um modelo teórico para calcular a complexidade ciclomática, denotado por *V(G)*, de um grafo de fluxo de um programa. Sua

métrica é um exemplo de uma métrica de controle de fluxo. *V(G)* pode ser computada pela equação 2.9:

$$V(G) = e - n + 2p \tag{2.7}$$

onde 2 é o "resultado de se adicionar uma aresta extra a partir do nó de saída ao nó de entrada de cada módulo componente do grafo" (HENDERSON-SELLERS; TEGARDEN, 1993).

O grafo de fluxo de controle de um segmento de um código é utilizado para mostrar seu fluxo de controle, onde os nós representam as tarefas de processamento e as arestas representam o fluxo de controle entre os nós (TIWARI; KUMAR, 2014).

De uma forma geral, o valor da complexidade ciclomática define um limite superior para a quantidade de testes necessários para cobrir todos os caminhos decisórios no código em questão. Esse é um limite superior já que nem todos os caminhos são necessariamente realizáveis (FERRAZ, 2008).

Diante do exposto, pode-se inferir que quanto menor a complexidade, menor a quantidade de testes necessários para o método em questão. Esse fato implica em outro: a quebra de um método em vários reduz a complexidade dos métodos mas aumenta a complexidade geral do código e, de forma geral, mantém a testabilidade do programa completo no mesmo nível (FERRAZ, 2008).

Já que a complexidade é um valor específico, é possível extrair da mesma uma referência. Baseado no trabalho de McCabe (1976), esses valores de referência são:

- 1-10, métodos simples, sem muito risco;
- 11-20, métodos medianamente complexos, com risco moderado;
- 21-50, métodos complexos, com risco alto, e
- 51 ou mais, métodos instáveis de altíssimo risco.

2.4 Engenharia de Software

2.4.1 Padrão de Projeto: State

O padrão *State* é um dos vinte e três padrões GoF (*Gang of Four* (GAMMA et al., 2000), destacado como um padrão comportamental que se preocupa com os algoritmos e as atribuições de responsabilidades entre objetos. O *State* permite que um objeto mude seu comportamento em tempo real. Durante sua execução, e dependendo do estado interno do objeto, esse padrão permite lidar com alterações comportamentais, dando uma

falsa sensação de o objeto ter mudado de classe. Esse padrão é utilizado quando o comportamento do objeto depende do seu estado ou quando existirem operações condicionais muito grandes (SALES, 2008).

A base do *State* é uma classe abstrata que representará os estados em comum. Subclasses, filhas dessa classe abstrata, sobrescrevem os comportamentos com detalhamento mais específico. (SALES, 2008).

A seguir, na Figura 11, será apresentado um exemplo de aplicação do padrão *State* elaborado por GAMMA et al. (2000).

Considere a classe TCPConnection que representa uma conexão numa rede de comunicações. Um objeto TCPConnection pode estar em diversos estados diferentes: Established (Estabelecida), Listening (Escutando) e Closed (Fechada). Quando um objeto TCPConnection recebe solicitações de outros objetos, ele responde de maneira diferente dependendo do seu estado corrente. Por exemplo, o efeito de uma solicitação de Open (Abrir), depende de se a conexão está no seu estado Closed ou no seu estado Estabished. O padrão State descreve como TCPConnection pode exibir um comportamento diferente em cada estado.

A ideia chave deste padrão é introduzir uma classe abstrata chamada TCPState para representar os estados da conexão na rede. A classe TCPState declara uma interface comum para todas as classes que representam diferentes estados operacionais. As subclasses de TCPState implementam comportamentos específicos ao estado. Por exemplo, as classes TCPEstablished e TCPClosed implementam comportamentos específicos aos estados Established e Closed de TCPConnection.

A Figura 11 ilustra esse raciocínio em um diagrama de classes.

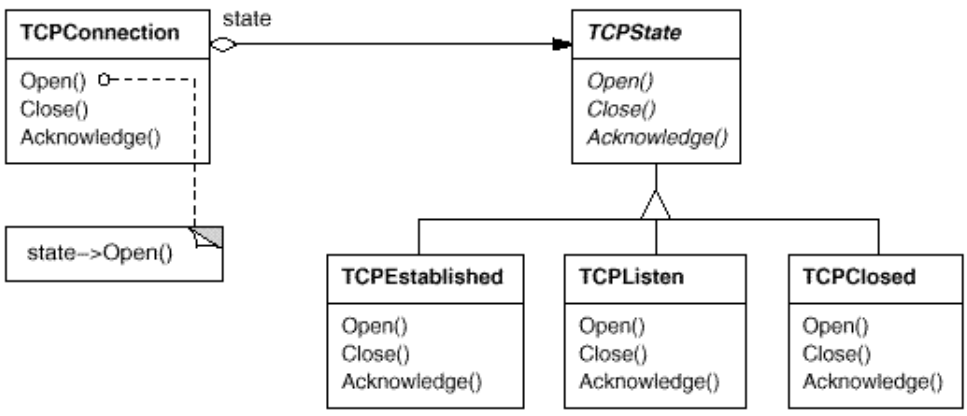

Figura 11: Diagrama de Classes. (GAMMA et al., 2000)

A classe TCPConnection mantém um objeto de estado (uma instância da subclasse de TCPState) que representa o estado corrente na conexão TCP.

Connection delega todas as solicitações específicas de estados para este objeto de estado. TCPConnection usa sua instância da subclasse de TCPState para executar operações específicas ao estado da conexão.

Sempre que a conexão muda de estado, o objeto TCPConnection muda o objeto de estado que ele utiliza. Por exemplo, quando a conexão passa do estado Established para o estado Closed, TCPConnection subtituirá sua instância de TCPEstablished por uma instância de TCPClosed.

Esse Padrão de Projeto foi utilizado da máquina de estados do jogo, sendo melhor explorado no capítulo Proposta.

2.4.2 Scrum

O *framework* Scrum consiste nos times do Scrum associadas a papéis, eventos, artefatos e regras. Cada componente dentro do *framework* serve a um propósito específico e é essencial para o uso e sucesso do Scrum. O Scrum prescreve quatro eventos formais, contidos dentro dos limites da *Sprint*, para inspeção e adaptação, como descrito na seção Eventos do Scrum deste documento (SCHWABER; SUTHERLAND, 2013).

- Reunião de planejamento da *Sprint*;
- Reunião diária;
- Reunião de revisão da Sprint, e
- Retrospectiva da Sprint.

O Product Owner, ou dono do produto, é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Times Scrum e indivíduos (SCHWABER; SUTHERLAND, 2013).

O coração do Scrum é a *Sprint*, um *time-boxed* de um mês ou menos, durante o qual um "Pronto", versão incremental potencialmente utilizável do produto, é criado. Sprints tem durações coerentes em todo o esforço de desenvolvimento. Uma nova *Sprint* inicia imediatamente após a conclusão da *Sprint* anterior. Também existe a Retrospectiva da Sprint, que é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima *Sprint*. A Retrospectiva da Sprint ocorre depois da Revisão da Sprint e antes da reunião de planejamento da próxima Sprint. Esta é uma reunião time-boxed de três horas para uma Sprint de um mês. Para Sprint menores, este evento é usualmente menor (SCHWABER; SUTHERLAND, 2013).

O Backlog do Produto (ou *Product Backlog*) é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. O Product Owner é responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e ordenação. A meta da *Sprint* é um objetivo definido para a Sprint que pode ser satisfeito através da implementação do Backlog do Produto (SCHWABER; SUTHERLAND, 2013).

Para esse trabalho, apenas um desenvolvedor trabalhou na implementação de todos os algoritmos de IA dos jogadores automatizados. Portanto, utilizou-se uma adaptação do *Scrum*, onde apenas o *Product Backlog* foi alimentado, e o tempo de desenvolvimento foi dividido em *Sprints*.

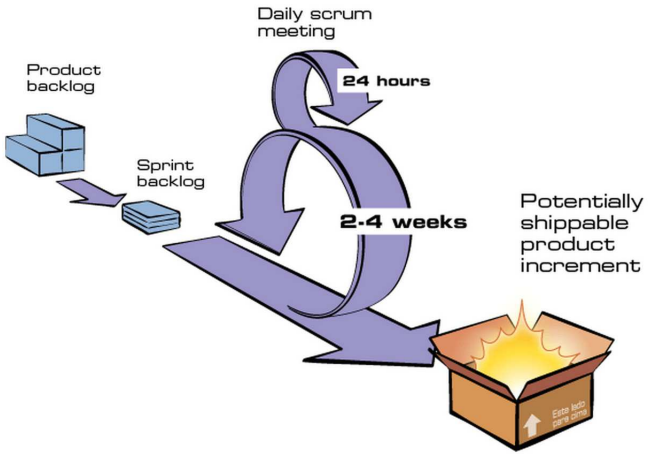

Figura 12: Representação da Metodologia Scrum

2.5 Considerações Finais

Para se desenvolver a Inteligência Artificial de jogadores autônomos, conhecer algumas técnicas tradicionais de Inteligência Artificial. Neste trabalho, utilizou-se os conceitos Padrão de Projeto *State*, Máquinas de Estado, Comportamentos de Direção, Grafos, Algoritmo A*, entre outros.

s

3 Suporte Tecnológico

A seguir, serão descritas as principais ferramentas, com as quais se conduziu o desenvolvimento desse trabalho. Esse suporte está orientado ao ambiente de desenvolvimento iOS, especificamente voltado ao domínio de jogos. Também foram utilizadas ferramentas para lidar com controle de versão, análise estática, entre outros.

O suporte tecnológico descrito nesse capítulo originou-se de estudos embasados na literatura.

3.1 Ferramentas de Desenvolvimento

3.1.1 Xcode

Xcode[1] é uma IDE (*Integrated Development Environment*) que contém um conjunto de ferramentas de desenvolvimento de software. Desenvolvida pela empresa Apple.Inc [2], essa ferramenta IDE apoia o desenvolvimento de software para sistemas OS X e iOS. Lançada pela primeira vez em 2003, a última versão estável está disponível na Mac App Store gratuitamente para usuários do sistema OS X Yosemite. Programadores que possuem uma conta desenvolvedor da Apple, podem baixar os pré-lançamentos de versões da ferramenta e versões anteriores através do domínio *Apple Developer*. Para este trabalho, foi utilizada a versão 7.1 do Xcode.

3.1.2 Sprite Kit

Sprite Kit é uma biblioteca de desenvolvimento de jogos para sistemas iOS que fornece uma infra-estrutura de renderização de gráficos e de animação que pode ser usada para animar imagens com texturas abritárias ou *sprites*. Sprite Kit utiliza um *loop* de renderização tradicional, onde o conteúdo de cada quadro é processado antes do mesmo ser renderizado. O jogo determina o conteúdo da cena e como esses conteúdos mudam em cada quadro. Sprite Kit faz o trabalho de renderizar os quadros de animação de forma eficiente utilizando *hardware* de vídeo. Sprite Kit é otimizado de forma que as posições dos *sprites* podem ser alteradas arbitrariamente em cada quadro da animação (APPLE, 2014).

Sprite Kit também fornece outras funcionalidades que são úteis para jogos, incluindo suporte para reprodução de som básico e simulação de física. Além disso, Xcode

[1] <https://itunes.apple.com/us/app/xcode/id497799835>
[2] <http://www.apple.com>

fornece suporte embutido para Sprite Kit de modo que efeitos especiais complexos e texturas em *atlas* podem ser criadas diretamente na IDE(APPLE, 2014). Para este trabalho, foi utilizada a versão do Sprite Kit que acompanha o Xcode 7.1.

3.1.3 Git

Git[3] é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade. O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos. Para este trabalho, foi utilizada a versão 2.4.2 do Git.

3.1.4 Bitbucket

Bitbucket[4] é um serviço de hospedagem de projetos controlados através do Mercurial e Git, sistemas de controle de versões distribuído. Bitbucket possui um serviço grátis e um comercial, sendo escrito em Python.

3.1.5 LaTeX

LaTeX[5] é um sistema de preparação de documentos para a composição tipográfica de alta qualidade. É mais frequentemente usado para documentos técnicos ou científicos de médio a grande porte, mas pode ser usado para quase qualquer forma de publicação. LaTeX não é um processador de texto. No caso, LaTeX incentiva os autores a não se preocuparem muito com a aparência de seus documentos, e sim em elaborar o conteúdo certo.

LaTeX é baseada na linguagem TeX de Donald E. Knuth. LaTeX foi desenvolvido pela primeira vez em 1985 por Leslie Lamport, e agora está sendo mantido e desenvolvido pelo Projeto LaTeX3 , disponível gratuitamente. Para este trabalho, foi utilizada a versão LaTeX2e.

3.1.6 XClarify

XClarify [6] é uma ferramenta de análise estática que simplifica o gerenciamento de um código complexo escrito em Objective-C. Através dela, arquitetos e desenvolvedores de *software* pode analizar a estrutura do código, especificar regras de *design*, realizar revisões efetivas no código e comparar a evolução da complexidade de diferentes versões do *software*. XClarify provê vários *feedbacks* do código, entre eles:

[3] <https://git-scm.com/>
[4] <https://bitbucket.org/>
[5] <http://latex-project.org/intro.html>
[6] <http://www.codergears.com/xclarify/faq>

- Dependências;
- Complexidade Ciclomática;
- Problemas de estrutura;
- Linhas de código, e
- Tamanho de métodos.

Para este trabalho, foi utilizada a versão 3.1.0 do XClarify.

3.2 Considerações Finais

Para o desenvolvimento de qualquer trabalho, é de suma importância definir quais ferramentas serão utilizadas a fim de se chegar ao resultado previsto. No decorrer deste trabalho, foram amplamente utilizadas ferramentas de escrita de documentos, modelagem de processos, escrita de código e análise do código.

4 Metodologia

Neste capítulo serão descritos os passos metodológicos que orientaram o desenvolvimento deste Trabalho de Conclusão de Curso, com detalhamento das atividades desde o levantamento bibliográfico até a escrita do TCC.

4.1 Pesquisa Exploratória

A pesquisa exploratória é realizada, segundo Vergara (2000), em áreas em que existe pouco conhecimento acumulado e sistematizado. É adequada quando se deseja aumentar o conhecimento sobre um dado assunto, ou, nas palavras de Gonçalves et al. (2004, p. 37), é "realizada para descobrir ou descrever melhor o(s) problema(s)-raiz que são apontados através de sintomas (ou queixas) para se alcançar os objetivos." Jr. et al. (2005) afirmam que a pesquisa exploratória é útil para o pesquisador que desconhece sobre o domínio investigado.

LAKATOS e MARCONI (2001) consideram que a pesquisa exploratória deve estar voltada para a formulação de questões ou de problemas de investigação, que aumentem a familiaridade do pesquisador com o assunto, o desenvolvimento de hipóteses sobre o tema pesquisado e a modificação e esclarecimento de conceitos. DENCKER (2001) observa que as pesquisas exploratórias utilizam grande quantidade de dados extraídos de fontes secundárias, estudos de casos selecionados e de observações informais, sendo os meios mais comuns de pesquisa exploratória a pesquisa bibliográfica e o estudo de caso. Para Samara e Barros (2007), a pesquisa exploratória tem como principais características a informalidade, a flexibilidade e a criatividade, permitindo um primeiro contato com a realidade a ser investigada.

4.2 Pesquisa Quantitativa

A pesquisa quantitativa vem da tradição das ciências naturais, onde as variáveis observadas são poucas, objetivas e medidas em escalas numéricas. Filosoficamente, a pesquisa quantitativa baseia-se numa visão dita positivista, onde, de acordo com Wainer (2007), tem-se:

- As variáveis a serem observadas são consideradas objetivas, isto é, diferentes observadores obterão os mesmos resultados em observações distintas;
- Não há desacordo do que é melhor e o que é pior para os valores dessas variáveis objetivas;

- Medições numéricas são consideradas mais ricas que descrições verbais, pois elas se adequam à manipulação estatística.

4.3 Pesquisa Qualitativa

A pesquisa qualitativa baseia-se na observação cuidadosa dos ambientes onde o sistema está sendo usado ou onde será usado, do entendimento das várias perspectivas dos usuários ou potenciais usuários do sistema, dentre outros aspectos. Dentre os métodos qualitativos que podem ser utilizados, de acordo com Wainer (2007), destacam-se:

- Estudos qualitativos observacionais;
- Pesquisa-ação (ou estudos qualitativos intervencionistas), e
- Outras formas de avaliação qualitativa.

4.4 Pesquisa-Ação

É importante que se reconheça a pesquisa-ação como um dos inúmeros tipos de investigação, que é um termo genérico para qualquer processo que siga um ciclo no qual se aprimora a prática pela oscilação sistemática entre agir no campo da prática e investigar a respeito dela. Planeja-se, implementa-se, descreve-se e avalia-se uma mudança para melhorar sua prática, aprendendo mais, no correr do processo, tanto a respeito da prática quanto da própria investigação. A Figura 13 mostra a representação em quatro fases do ciclo básico da investigação-ação (TRIPP, 2005).

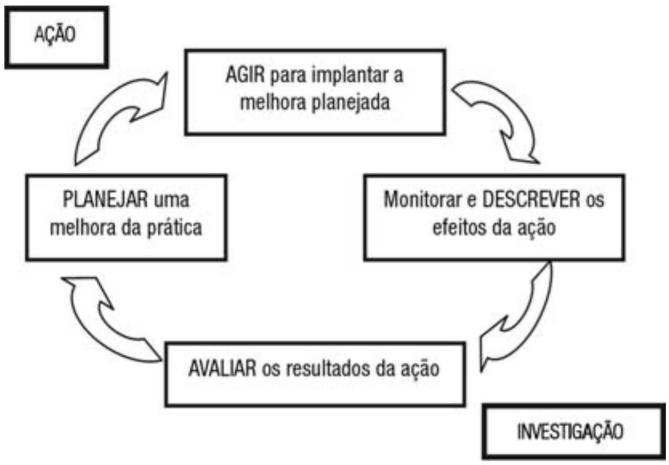

Figura 13: Representação da Investigação-Ação

A pesquisa-ação é uma forma de investigação-ação que utiliza técnicas de pesquisa consagradas para informar a ação que se decide tomar para melhorar a prática. Isso

posto, embora a pesquisa-ação tenda a ser pragmática, ela se distingue claramente da prática e, embora seja pesquisa, também se distingue claramente da pesquisa científica tradicional, principalmente porque a pesquisa-ação ao mesmo tempo altera o que está sendo pesquisado e é limitada pelo contexto e pela ética da prática (TRIPP, 2005).

4.5 Planejamento dos Passos Metodológicos

No início da escrita do TCC, a pesquisa exploratória foi bastante cabível, pois houve um estudo aprofundado de técnicas e algoritmos relacionados aos jogadores autônomos, e o conhecimento obtido foi documentado.

O fluxo ilustrado na Figura 14 está organizado quanto as atividades realizadas ao longo deste trabalho. Essa modelagem foi definda na ferramenta Bizagi[1].

Inicialmente, foi feito um levantamento bibliográfico de várias abordagens de IA para jogadores automatizados visando proporcionar maior familiaridade com o problema, analisando algoritmos clássicos que se enquadram nesse tema. Analisando-se vários exemplos na literatura e em artigos científicos, definiu-se que a implementação dos jogadores automatizados seria feita seguindo a estratégia de Máquinas de Estados, Comportamentos de Direção, Grafo e Algoritmos de Busca de Menor Caminho. Essas estratégias foram escolhidas visando conferir uma modelagem bem como uma implementação preliminar ao tema abordado nesse projeto, em atendimento aos objetivos do trabalho, ou seja, procurando lidar com a questão de pesquisa (i.e. ter jogadores automatizados capazes de jogar como se fossem jogadores humanos, adicionalmente, procurando equilibrar o jogo.)

Após o processo de escolha da estratégia, a proposta foi elaborada, o suporte tecnológico definido e o referencial teórico refinado. Tais ações são representadas pelas atividades Elaborar Proposta e Levantar Suporte Tecnológico e pelo artefato Referencial Teórico Refinado, descritas na Figura 14. Estes passos forneceram base para o Desenvolvimento da Prova de Conceito, onde a arquitetura da Inteligência Artificial dos *bots* foi desenvolvida. Terminada a Prova de Conceito, o Trabalho de Conclusão de Curso (TCC) 1 foi escrito. Posteriormente, o mesmo foi apresentado à banca avaliadora. As sugestões da banca foram coletadas, e as devidas correções foram realizadas.

As atividades de desenvolvimento foram guiadas por uma adaptação da metodologia ágil *Scrum*. Para isso, o *Product Backlog* foi alimentado com diversas Histórias de Usuário. As Histórias de Usuário escritas tiveram que ser adaptadas para se adequarem mais precisamente ao contexto de jogo.

[1] <http://www.bizagi.com/>

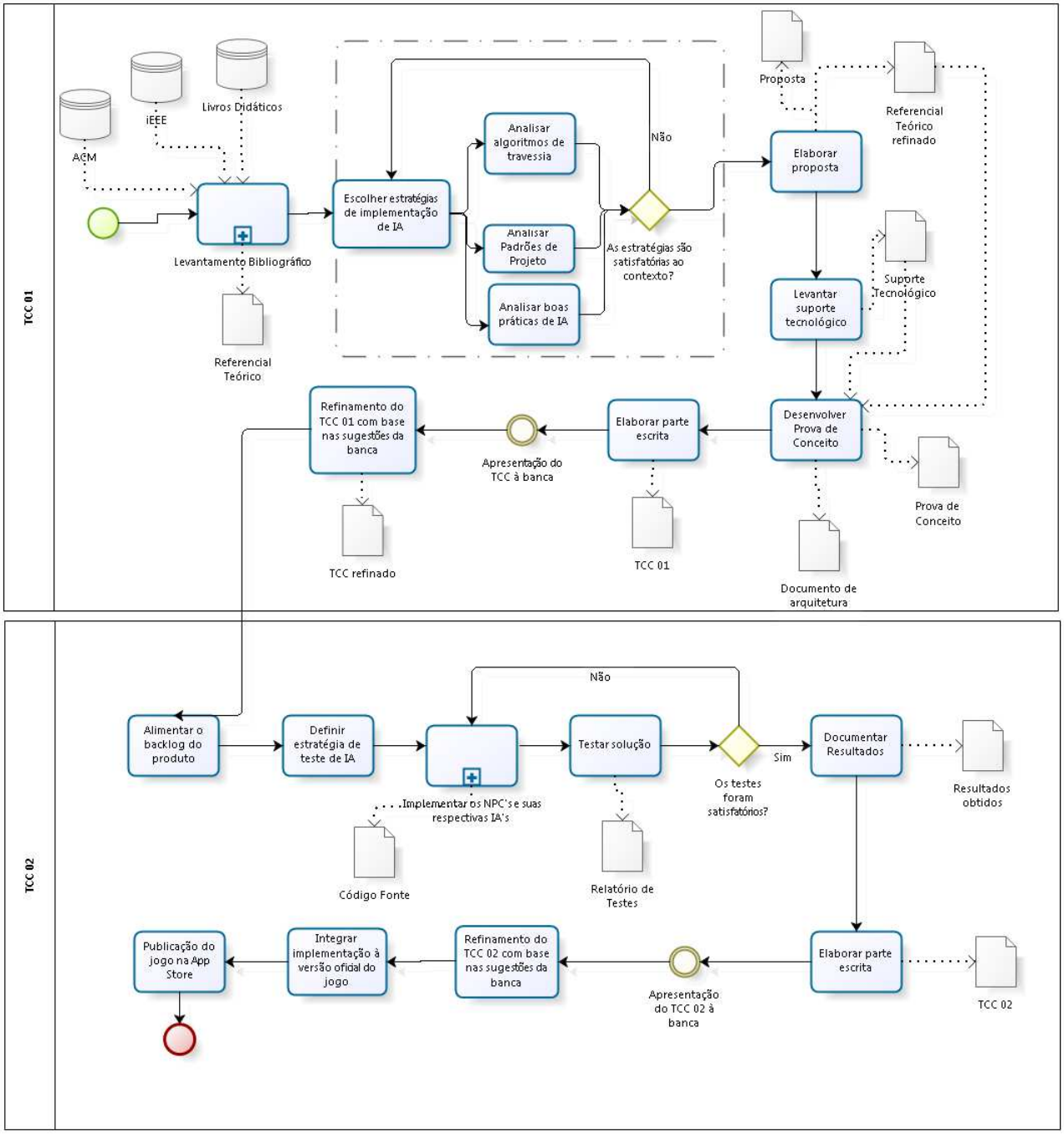

Figura 14: Processo do TCC

Os tipos de testes foram definidos, e então o trabalho de codificação dos algoritmos teve início. Enquanto os testes não fossem executados com sucesso, o código era reescrito e melhorado.

Como os NPCs precisavam possuir um nível alto de qualidade no que diz respeito

à inteligência (precisam ter um comportamento similar ao de jogadores humanos), foram definidos dois tipos de testes para o trabalho implementado: testes unitários e testes com usuários. Para os testes unitários, foi utilizada a biblioteca *XCTest* [2], que é nativa da linguagem *Objective-C*. Para testes com usuários, o objetivo era coletar um *feedback* real e expressivo do comportamento dos *bots*, e em seguida, realizar os ajustes necessários segundo as observações e respostas adquiridas junto aos usuários.

Após o código-fonte ser escrito e testado, os resultados dos testes foram analisados a fim de que fossem documentados de maneira apropriada. A análise de resultados foi orientada por uma abordagem qualitativa e quantitativa. A primeira, qualitativa, realizada com base na satisfação e percepção dos potenciais usuários finais do jogo. Já a segunda, quantitativa, com análise estática de código e cobertura de teste. A análise estática foi feita pela ferramenta XClarify [3], onde a complexidade ciclomática foi calculada e ajustada através de refatorações do código.

A integração dos NPCs com o jogo oficial e a publicação do jogo da App Store correspondem às atividades previstas na última fase de todo o ciclo de realização do trabalho proposto.

A seguir, será explicado como as Histórias de Usuário foram escritas no contexto desse trabalho, e o *Product Backlog* será apresentado.

4.6 Histórias de Usuário

Uma História de Usuário é uma breve descrição de uma necessidade, uma característica ou um desejo do ponto de vista de um usuário específico em um projeto de software. Pode também conter uma explicação de sua importância, seus benefícios ao projeto e critérios para conclusão. A estrutura básica de uma História de Usuário pode ser como se segue (SCHETINGER et al., 2011):

> "Como um (usuário específico) desejo (algo) para que (benefício)."

Keith (2010) recomenda o uso de Histórias de Usuário no desenvolvimento de jogos, não apenas para sua aplicação em metodologias ágeis, mas para que adquiram significado especial no contexto de jogos. Utilizando-se da definição de História de Usuário descrita anteriormente, o seguinte exemplo pode ser sugerido: "Como um jogador, desejo jogar utilizando um *joystick*". O jogador é o usuário final do jogo, e nessa História de Usuário é exigido um suporte para um dispositivo de entrada específico (SCHETINGER et al.,

[2] <https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/03-testing_basics.html>

[3] http://www.codergears.com/xclarify/

2011).

Como regra geral, um jogo tenta simular um determinado cenário para provocar a imersão dos jogadores, independente do gênero. Criar Histórias de Usuário para estes cenários diferem do uso comum em projetos de software, onde um caso de uso simples e direto é escrito baseado nas necessidades dos usuários do sistema, como no exemplo descrito no parágrafo anterior. O seguinte exemplo demonstra uma abordagem diferente na escrita de uma História de Usuário para jogos:

"Como um médico, desejo ser capaz de visualizar a saúde dos meus aliados, porque seria mais fácil de ajudar jogadores feridos."

Neste exemplo criado por Schetinger et al. (2011), o usuário final pertence ao contexto do jogo de modo a representar uma característica específica do mesmo, descrevendo melhor o cenário que o jogo tenta alcançar. Outro exemplo pode ser visto a seguir.

"Como uma unidade inimiga, desejo ser capaz de roubar itens de jogadores."

Nessa História um jogador automatizado deseja ser capaz de roubar itens de outros jogadores. Como é possível que esse inimigo, não sendo um usuário final do jogo, possa fazer pedidos? Como explicitado anteriormente, as Histórias de Usuário no contexto de jogos geralmente extrapolam sua expressividade para criar meios de descrever uma funcionalidade. O autor da História poderia ser o *designer* da IA dos jogadores automatizados, visando criar novos comportamentos para inimigos, enquanto necessita da ajuda de alguém para implementar essas mudanças no projeto. Se este for o caso, então essa História de Usuário iria primeiramente: comunicar um pedido a diferentes membros da equipe, descrever um novo aspecto do jogo, e, dependendo do contexto do time de desenvolvimento, a História se tornaria uma tarefa no planejamento (SCHETINGER et al., 2011).

O *Product Backlog* deste trabalho pode ser observado na Tabela 1. As Histórias de Usuário foram escritas seguindo o padrão de Schetinger et al. (2011), onde o *bot* representa o usuário final com necessidades e desejos a fim de que uma funcionalidade específica do trabalho seja entendida de forma mais adequada.

Atendendo cada História de Usuário por vez, todas as funcionalidades propostas foram desenvolvidas.

Identificador	História de Usuário
1	Eu, como jogador automatizado, desejo me movimentar para que possa atingir outros lugares no mapa.
2	Eu, como jogador automatizado, desejo seguir rotas pré-definidas para que possa me movimentar de forma sistemática e ordenada pelo mapa.
3	Eu, como jogador automatizado, desejo andar pelo mapa sem me chocar nos obstáculos para que eu consiga andar pelo mapa sem impedimentos.
4	Eu, como jogador automatizado, desejo seguir as rotas pré-definidas utilizando de Comportamentos Direcionais para que possa me movimentar de forma mais realista.
5	Eu, como jogador automatizado, desejo andar de um ponto a outro no mapa pelo menor menor caminho para que eu possa chegar em um ponto desejado com menos tempo possível.
6	Eu, como jogador automatizado, desejo ver outros jogadores automatizados no mapa para que eu os reconheça como aliados ou inimigos.
7	Eu, como jogador automatizado, desejo atirar para que possa causar dano em um inimigo.
8	Eu, como jogador automatizado, desejo me locomover até a base inimiga para que possa roubar a gema.
9	Eu, como jogador automatizado, desejo mudar de estado de ação conforme certas condições para que eu possa tomar uma decisão mais cabível para determinada situação.
10	Eu, como jogador automatizado, desejo perseguir um inimigo para que eu possa atacá-lo de forma mais realista.
11	Eu, como jogador automatizado, desejo levar a gema inimiga para minha base aliada para que possa ganhar o jogo.
12	Eu, como jogador automatizado, desejo patrulhar minha base aliada para impedir que inimigos roubem a gema aliada.
13	Eu, como jogador automatizado, desejo possuir um perfil atacante para que possa roubar a gema inimiga.
14	Eu, como jogador automatizado, desejo possuir um perfil defensor para que possa defender a gem aliada.

Tabela 1: Histórias de Usuário para Implementação de *Bots*

Atividades	Março	Abril	Maio	Junho	Julho
Levantar Material Bibliográfico	X	X	X		
Escolher Estratégias de Implementação de IA		X	X		
Elaborar Proposta		X	X	X	
Desenvolver Prova de Conceito			X	X	
Elaborar Parte Escrita do TCC 01			X	X	
Apresentar TCC 01 à banca					X
Refinar TCC 01 conforme sugestões da banca					X

Tabela 2: Cronograma do TCC 01

Atividades	Agosto	Setembro	Outubro	Novembro	Dezembro
Alimentar o *Backlog do Produto*	X				
Definir Estratégia de Teste de IA	X				
Implementar *bots*	X	X	X		
Testar solução		X	X		
Documentar Resultados			X	X	
Elaborar parte escrita do TCC 02			X	X	
Apresentar TCC 02 à banca				X	
Refinar TCC 02					X
Integrar implementação ao jogo					X
Publicar jogo na *App Store*					X

Tabela 3: Cronograma do TCC 02

4.7 Cronograma

A Tabela 2 e Tabela 3 apresentam os cronogramas que foram utilizados para, respectivamente, a realização do TCC_1 e a realização do TCC_2.

4.8 Considerações Finais

Uma metodologia de desenvolvimento bem definida é essencial para o sucesso de qualquer trabalho. Vários conceitos de Engenharia de Software foram utilizados no decorrer do desenvolvimento deste trabalho, tais como: metodologia de pesquisa científica (ex. exploratória, quantitativa e qualitativa), metodologia de desenvolvimento (ex. adaptação do Scrum), metodologia de análise de resultados (ex. pesquisa quantitativa e qualitativa), planejamento de prazo, verificação e validação, entre outros.

5 Prova de Conceito

Para que as técnicas e abordagens que dão suporte à Inteligência Artificial dos *bots* fossem compreendidas de forma mais adequada (no início do trabalho), desenvolveu-se uma prova de conceito. Essa prova de conceito permitiu a obtenção de um melhor entendimento das estratégias de implementação escolhidas para o desenvolvimento da aplicação. Este capítulo também visa exemplificar e justificar as escolhas das estratégias adotadas para a implementação dos jogadores automatizados.

5.1 Representação do Mapa em Grafos

Primeiramente, analisou-se o problema de como gerar caminhos e pontos no mapa a fim de que os jogadores automatizados possam se movimentar seguindo esses pontos. Para isso, considerou-se a estratégia de *Grids*(RABIN, 2002, p. 560) e de grafos para a representação do espaço onde o jogador pode se locomover. A abordagem de grafos foi escolhida, pois segundo Rabin (2002), enquanto uma típica célula de uma *Grid* é conectada com oito células vizinhas, conforme é vista na Figura 15, um nó de um grafo pode ser conectado a qualquer número de nós. Isso faz com que grafos de nós se tornem muito mais flexíveis do que *Grids*.

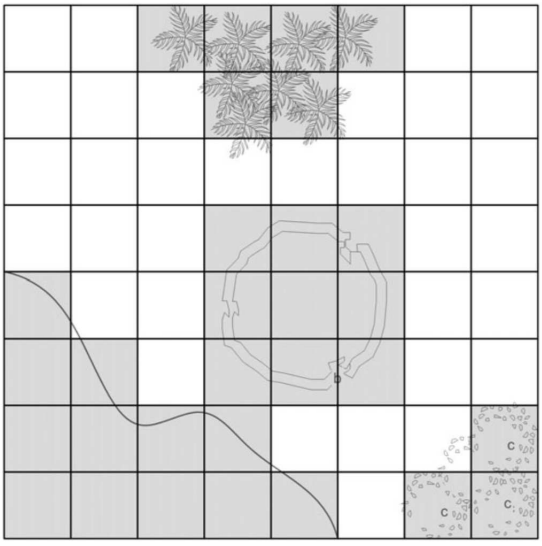

Figura 15: Exemplo de um Mapa Representado por *Grid*

Para a representação do grafo, deduziu-se uma malha com um número elevado de nós. A Figura 16 exemplifica essa representação, onde cada nó (círculos pretos) estaria ligado ao nó vizinho através de uma aresta invisível.

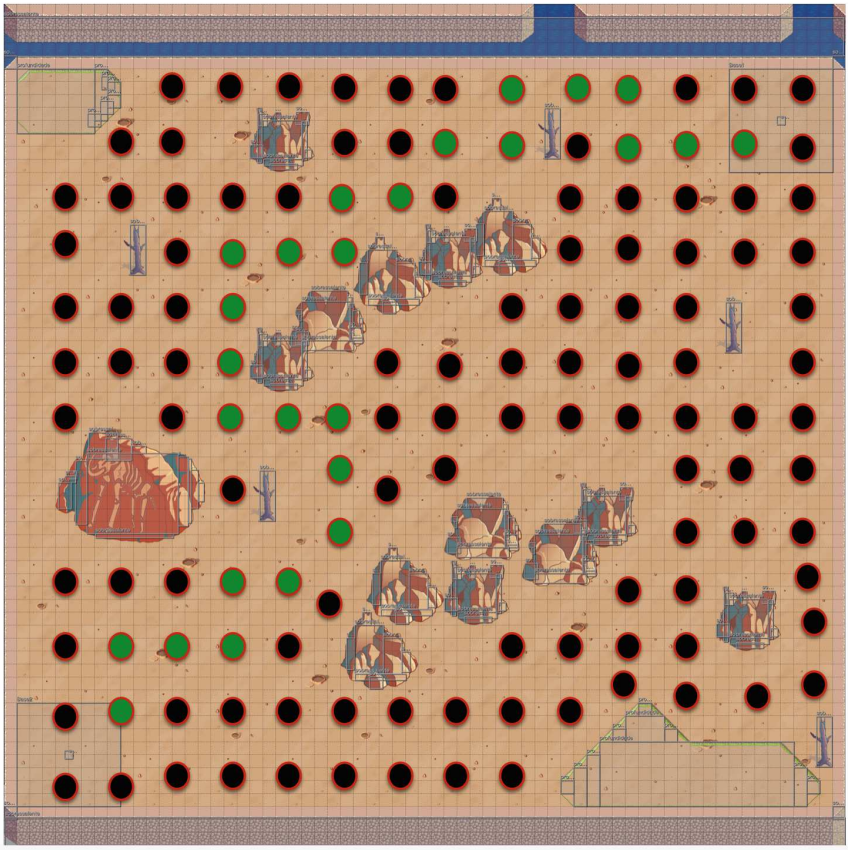

Figura 16: Representação Inicial do Mapa com Grafo

Nessa representação de mapa, considerou-se utilizar o algoritmo de Busca em Largura para se determinar o menor caminho entre dois nós dados. Tomou-se essa decisão pois esse algoritmo não faz uso da distância entre os nós (peso das arestas) para o cálculo do caminho. Entretanto, após se analisar outras estratégias e algoritmos (A* por exemplo), concluíu-se que essa representação de mapa demandaria uma quantidade excessiva e desnecessária de memória do *hardware*, pela grande quantidade de nós e arestas a serem gerados.

Com base nisso, desenvolveu-se uma nova representação do mapa com grafos, agora com menos nós e arestas. A Figura 19 (localizada no Capítulo 6) mostra essa abordagem. Nessa representação, as arestas possuem um peso, que é a distância em *pixels* que cada nó possui um do outro. Para busca de menor caminho, optou-se pelo algoritmo A*, já que os nós possuem pesos diferentes uns dos outros.

5.2 Comportamentos de Direção

Para validação dos Comportamentos de Direção, codificou-se em Objective-C três das estratégias criadas por Reynolds (1999): *Seek*, *Flee* e *Arrive*. O Apêndice T mostra o código implementado.

Os exemplos gerados demonstram um quadrado azul que possui uma tragetória fixa, e um quadrado vermelho que executa várias ações. A Figura 17 demonstra a aplicação sendo executada em um simulador de iPhone. As setas representam os vetores que influenciam a movimentação do quadrado, onde *Va* é o vetor da velocidade atual do quadrado, *Vd* é o vetor da velocidade desejada, e *Fd* é a força direcional calculada que efetivamente age sobre o quadrado vermelho.

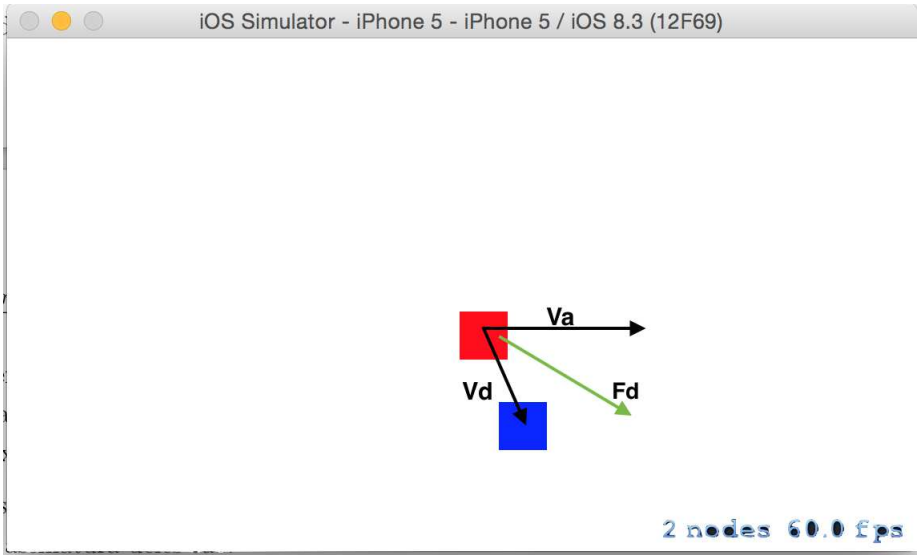

Figura 17: Aplicação Executando em um Simulador

A classe SteeringBehaviours.m possui três métodos que executam os algoritmos de direção. A assinatura deles são:

- `-(CGPoint) seek:(CGPoint)targetPos;`
- `-(CGPoint) flee:(CGPoint)targetPos`, e
- `-(CGPoint) arrive:(CGPoint)targetPos andDeceleration:(float)slowingRadius`.

O parâmetro `targetPos` é uma estrutura que possui duas variáveis (x,y) que representam a posição atual do quadrado azul em cada *frame* da simulação. O parâmetro `slowingRadius` representa a distância mínima exigida entre o quadrado azul e o vermelho para que este comece a desacelerar.

Cada método retorna uma vetor de força direcional que atua sobre o quadrado vermelho. Na simulação, pode-se perceber que os algoritmos implementados atendem bem ao contexto pretendido.

5.3 Considerações Finais

Percebeu-se que o desenvolvimento da prova de conceito foi de grande valia para a concretização das ideias e algoritmos propostos. Analisar diferentes algoritmos e abordagens também foi fundamental para que uma proposta coerente e embasada fosse escrita.

6 Desenvolvimento dos Jogadores Automatizados

Este capítulo visa exemplificar em detalhes todo o comportamento implementado para os jogadores automatizados no jogo *Lost Gems*. Sabe-se que os jogadores automatizados devem atingir os objetivos apresentados na Introdução desse documento.

O presente capítulo está organizado da seguinte forma: apresentação do mapa do jogo, o qual é percorrido pelos jogadores automatizados; descrição dos comportamentos secundários, os quais foram implementados procurando refinar os movimentos e a percepção dos jogadores automatizados visando aproximar do comportamento humano; e comportamentos dos jogadores automatizados, os quais foram modelados e implementados com base nos objetivos geral e específicos desse projeto.

Por fim, são descritas as considerações finais do capítulo.

6.1 Mapa do Jogo

Os jogadores automatizados devem percorrer o mapa do jogo com fluidez e sem se colidirem com os obstáculos. A Figura 18 ilustra o mapa que foi utilizado para desenvolver a Inteligência Artificial desses *bots*.

No canto inferior esquerdo e no canto superior direito ficam posicionadas as gemas que serão capturadas de ambas as equipes. A localização inicial das gemas é chamada de base. As bases também são as posições iniciais de todos os jogadores (humanos e automatizados) no início da partida. As árvores, as pedras e as naves espaciais são os obstáculos do mapa com as quais os jogadores não podem atravessar.

Para que os jogadores automatizados possam percorrer o mapa, é necessário que possuam um guia de quais posições devem seguir a fim de que posssam chegar até o seu objetivo. Para tanto, utilizou-se a estratégia de grafos no mapa, onde cada nó indica para qual posição no mapa o jogador automatizado deve se movimentar. A Figura 19 mostra uma representação de nós e arestas a qual foi implementada no mapa. Essa representação serve de base para a locomoção de todos os jogadores automatizados do jogo.

Os círculos pretos representam os nós, e as hastes pretas as aretas. Os números dentro dos nós têm o propósito de nomeá-los. Cada nó possui uma referência x,y que representa a sua posição no mapa em pontos cartesianos, e uma referência para os nós vizinhos. Cada aresta possui um atributo que indica a distância entre dois nós em termos de pontos (a biblioteca *Sprite Kit* define o tamanho desses pontos conforme as particu-

Figura 18: Mapa do Jogo

lariedades de cada dispositivo móvel). A Figura 19 servirá de referência para explicação dos algoritmos e de alguns comportamentos dos jogadores automatizados no jogo.

6.2 Comportamentos Secundários

Antes que os objetivos finais de comportamento dos *bots* fossem atacados diretamente, vários outros comportamentos e algoritmos foram implementados. Isso foi necessário a fim de que o movimento e a percepção dos jogadores automatizados, em relação ao jogo, fossem otimizados e também chegassem o mais próximo possível do comportamento humano. Para tanto, várias classes auxiliares (ou secundárias) foram escritas, conforme pode ser visto no diagrama de classes ilustrado na Figura 20.

A seguir, é descrita a função de cada classe presente na Figura 20.

- `Bot`: Classe responsável por conter todos os atributos e métodos necessários para o funcionamento de todas as características dos *bots*. O Apêndice A mostra o código fonte escrito para a classe. Os métodos de renderização, animação, configuração da barra de vida, entre outros métodos que não lidam diretamente com os algoritmos de inteligência, foram omitidos neste TCC.

- `Steering Behaviours`: Essa classe possui os métodos que realizam os cálculos das forças direcionais de cada comportamento de direção que os *bots* utilizam (Apêndice

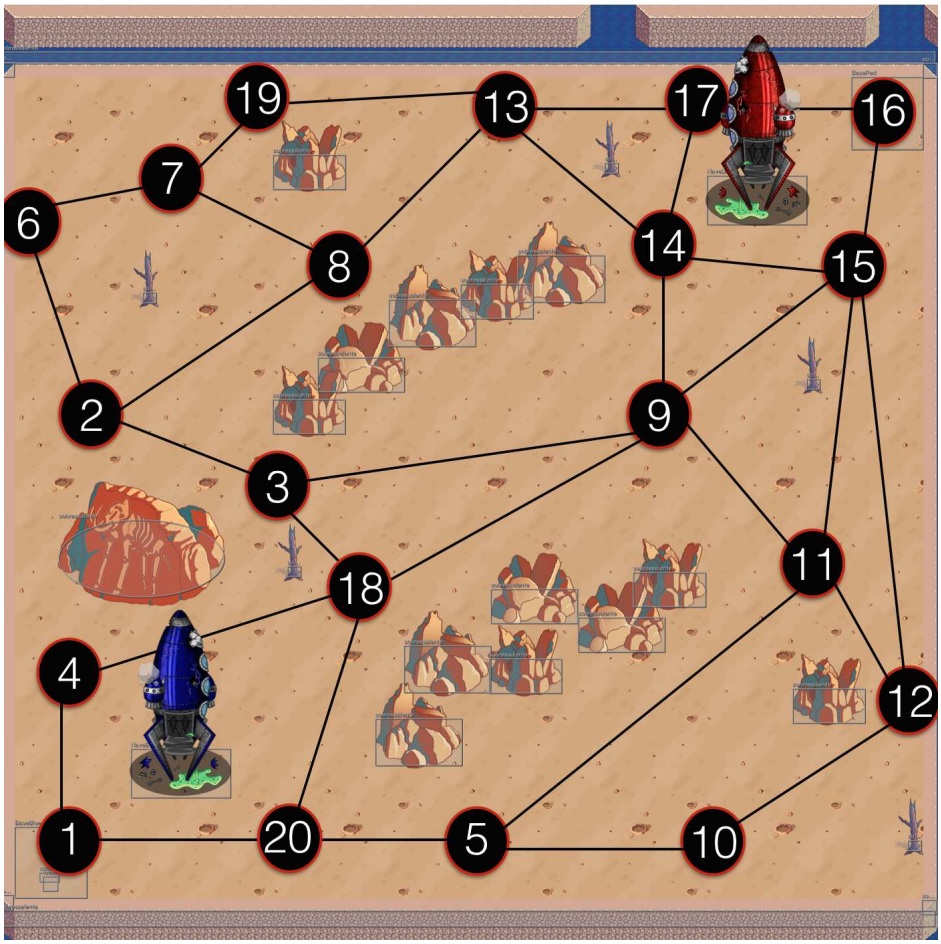

Figura 19: Mapa com Grafo

B). Esses métodos são chamados no *loop* principal do jogo para cada *bot*, de acordo com o seu estado atual. Os métodos são:

- `-(CGPoint) seek:(CGPoint)targetPos andBot:(Bot *)bot`: Calcula as forças correspondentes ao comportamento Seguir. O parâmetro `targetPos` é o vetor posição do alvo a ser seguido.
- `-(CGPoint) flee:(CGPoint)targetPos andBot:(Bot *)bot`: Calcula as forças correspondentes ao comportamento Fuga. O parâmetro `targetPos` é o vetor posição do alvo de onde o *bot* deve fugir.
- `-(CGPoint) followPath:(NSMutableArray *)points bot:(Bot *)bot`: Calcula as forças correpondentes ao comportamento Seguir Caminho. O parâmetro `points` correponde ao caminho de nós que o *bot* deverá trilhar, utilizando das forças direcionais do comportamento Seguir.
- `-(CGPoint) avoidanceCollision:(SKNode*) obstacle andBot:(Bot *)bot`: Calcula as forças correpondentes ao comportamento Evitar Obstáculos. O parâmetro `obstacle` refere-se ao obstáculo com o qual o *bot* colidiu.

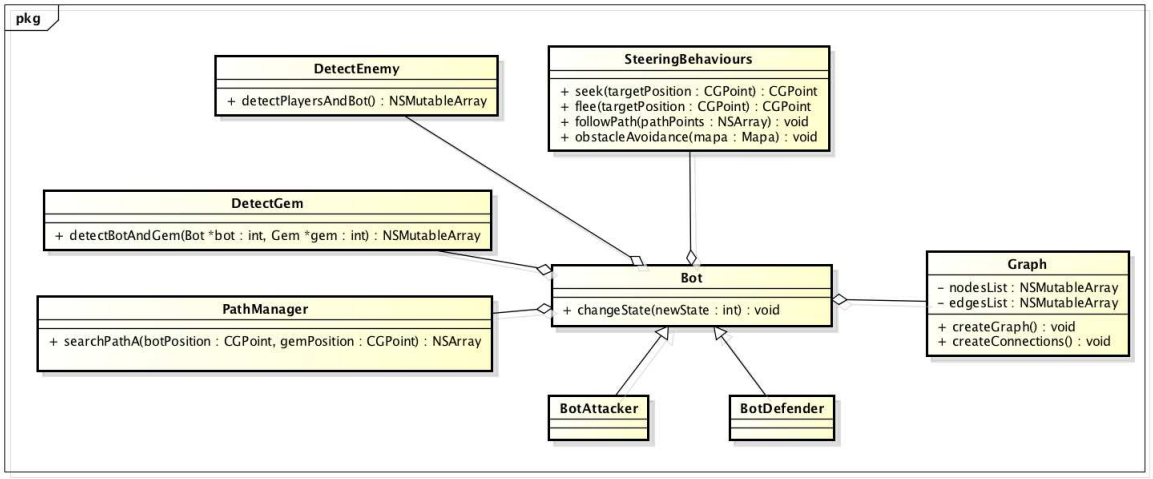

Figura 20: Diagrama de Classes Geral

- `Graph`: Tem o propósito de criar o grafo do mapa e suas respectivas conexões (Apêndice C). O método `createGraph` (linha 33) tem a função de ler dois arquivos do tipo *plist* (arquivo XML que provê uma estrutura dicionários para a linguagem *Objective-C*): um possui todos os números dos nós e suas respectivas posições *x, y*, e o outro aquivo possui as conexões de todos os nós. Isso é feito a fim de alocar os nós e as arestas nos *arrays* `nodesList` (lista com todos os nós) e *edgesList* (lista com os pesos das arestas). O peso das arestas é calculado com base na distância euclidiana dos nós vizinhos.

  Em seguida, o método `createConnections` é chamado para que os elementos do *array* `nodesList` sejam encadeados, criando assim uma lista de adjacência. A Figura 21 mostra o arquivo *plist GraphConnections* (responsável por conter os nós e seus respectivos vizinhos), e a Figura 22 ilustra o arquivo *plist GraphPositions* (responsável por conter os nós e suas respectivas posições *x, y* em relação ao mapa do jogo).

Key	Type	Value
▼ Root	Dictionary	(20 items)
1	String	4,20
2	String	3,6,8
3	String	2,9,18
4	String	1,18
5	String	20,10,11
6	String	2,7
7	String	6,8,19
8	String	2,7,13
9	String	3,11,14,15,18
10	String	5,12
11	String	5,9,12,15
12	String	10,11,15
13	String	8,14,17,19
14	String	9,13,15,17
15	String	9,11,12,14,16
16	String	17,15
17	String	13,14,16
18	String	3,4,9,20
19	String	7,13
20	String	1,5,18

Figura 21: Arquivo *plist* das Conexões dos Nós

Key	Type	Value
▼ Root	Dictionary	(20 items)
1	String	125,220
2	String	262,1227
3	String	631,1078
4	String	135,577
5	String	1063,255
6	String	116,1642
7	String	407,1728
8	String	735,1543
9	String	1410,1227
10	String	1716,432
11	String	1764,807
12	String	2000,506
13	String	1059,1851
14	String	1487,1579
15	String	1946,1418
16	String	1931,1893
17	String	1575,1865
18	String	867,831
19	String	557,1868
20	String	717,237

Figura 22: Arquivo *plist* das Posicões x,y dos Nós

- `DetectEnemy`: Essa classe tem a função de detectar inimigos próximos ao *bot* (Apêndice D). Esses inimigos são outros *bots* ou jogadores humanos que pertencem a um time diferente do jogador automatizado que utiliza essa classe. O método `detectPlayersAndBot` tem a função de verificar se o jogador humano está dentro de um determinado raio, e, em seguida, ocorre a iteração do *array* de *bots* para a verificação se cada jogador inimigo (i. e. jogador automatizado ou jogador humano) está dentro do "raio de visão" do *bot*. Em caso positivo, o método retorna a referência do jogador ou *bot*

inimigo. Se o *bot* estiver muito próximo de um aliado, ele chama o método *flee* da classe `Steering Behaviours` que lhe imprimirá uma força direcional contrária ao seu movimento. Isso é feito para que sua posição não se confunda com a do aliado.

- `PathManager`: O seu método principal, `searchPathbyRootandDestination`, implementa o algoritmo A* de menor caminho (Apêndice E). Ele recebe como parâmetros `rootNode`, que é o nó de partida, e `destination`, que representa o nó destino. O seu retorno é uma lista de nós que, lidos sequencialmente, representam o menor caminho pelo grafo de um nó a outro.

- `DetectGem`: Tem o papel de informar ao *bot* o menor caminho de nós até uma gema que se encontra fora de sua respectiva base (Apêndice F).

- `BotAttacker`: Representa a classe especializada da classe Bot que implementa o perfil atacante do *bot* (Apêndice G).

- `BotDefender`: Representa a classe especializada da classe Bot que implementa o perfil defensor do *bot* (Apêndice H). Tanto a classe BotAttacker quanto a BotDefender serão explicadas detalhadamente nas seções a seguir.

6.3 Comportamento dos Jogadores Automatizados

Sabe-se que os jogadores automatizados que atuam no jogo *Lost Gems* possuem uma série de comportamentos. Esses comportamentos devem ser realizados a fim de que os objetivos estabelecidos possam ser cumpridos de forma satisfatória. Para tanto, dividiu-se o comportamento dos jogadores automatizados em três macro perfis: atacante, defensor e seguidor.

6.3.1 Perfil Atacante

O jogador automatizado atacante é responsável por tomar a iniciativa de procurar a gema no mapa, capturá-la e levá-la até a base aliada para ganhar a partida. Também deve atacar os inimigos que encontrar em seu caminho, assim como persegui-los. Essa perseguição é temporária, acabando quando o jogador automatizado "morrer" ou derrotar o seu oponente. Após o fim da perseguição, o jogador automatizado deve continuar a sua busca pela gema, levar a gema à base aliada, ou proteger o portador da gema inimiga.

Esse comportamento foi deduzido de forma a se adequar a uma Máquina de Estados, onde cada comportamento é um estado da máquina. A Figura 23 indica um diagrama de estados elaborado para o perfil atacante de forma a facilitar a compreensão e a modelagem do comportamento do jogador automatizado.

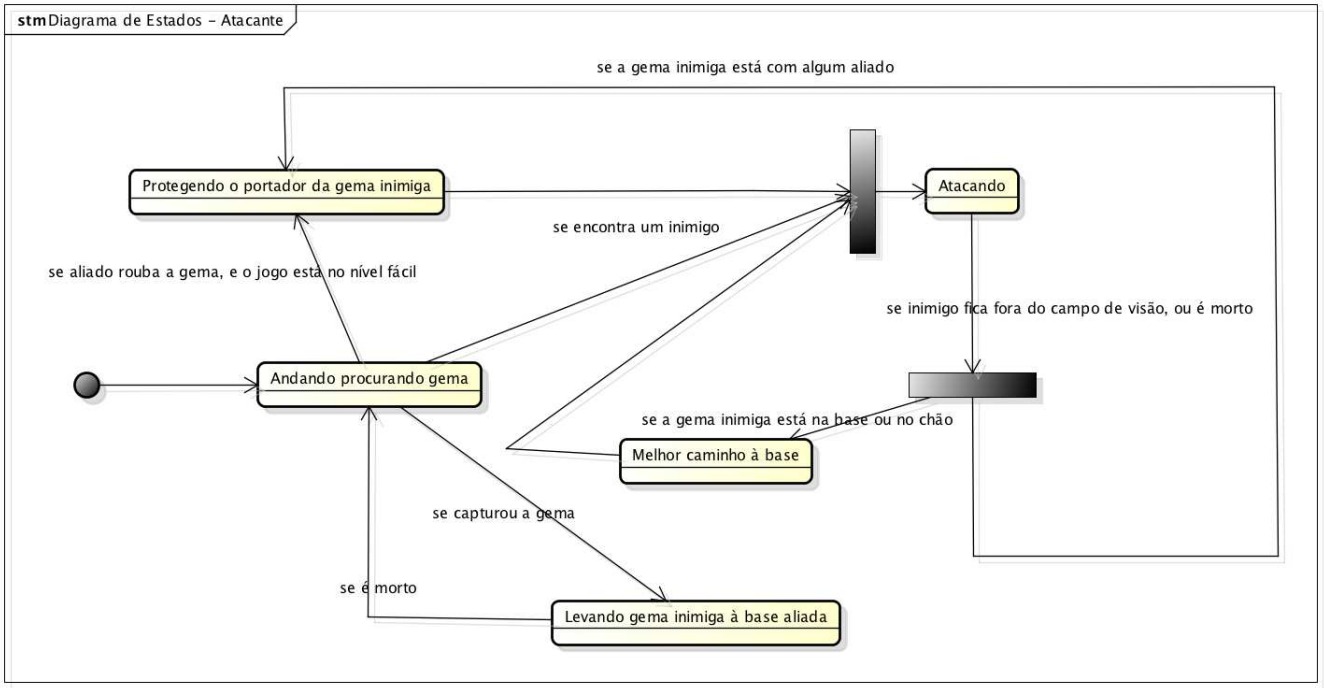

Figura 23: Diagrama de Estados - Perfil Atacante

A Figura 24 demonstra o diagrama de classes para o comportamento do jogador automatizado atacante.

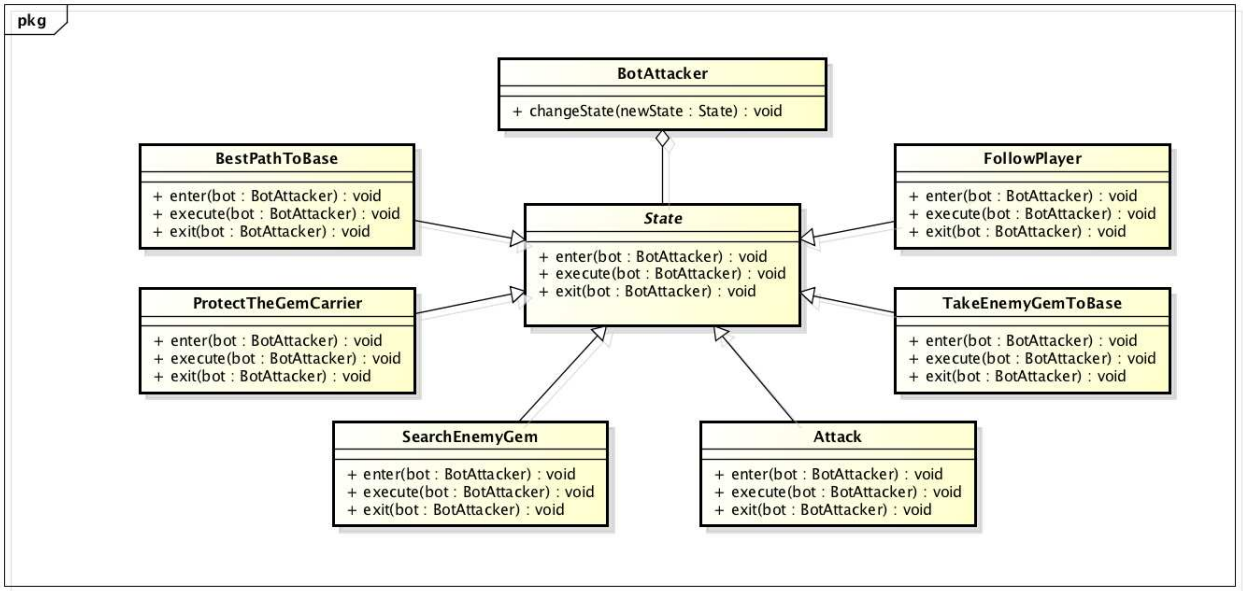

Figura 24: Diagrama de Classes - Perfil Atacante

Nota-se a utilização do Padrão de Projeto *State* para construir o diagrama. Nesse caso, cada estado do jogador é uma classe única, que herda de uma classa abstrata State, que por sua vez possui uma relação de agregação com a classe BotAttacker.

Cada classe que representa um estado possui pelo menos três métodos básicos: `enter`, `execute` e *exit*. O método `enter` é executado apenas quando o *bot* entra no novo estado. O método `execute` é chamado a cada *loop* principal do jogo e é responsável por executar as principais funções do estado. Também verifica se as condições para mudança de estado são necessárias. O método `exit`, por fim, realiza configurações necessárias quando um estado muda para outro (reinicialização de atributos, chamada de métodos, entre outros).

O código fonte da classe BotAttacker, conforme apresentado no Apêndice G, possui os seguintes métodos:

- `-(void) initActions`: Responsável por iniciar a máquina de estados, atribuindo uma instância da classe SearchEnemyGem ao atributo `currentState` (que guarda o estado atual do *bot*).
- `-(void) changeEstate:(id <StateBot>)newState`: Responsável por fazer a mudança de estados. Esse método é chamado nas classes que representam os estados do *bot*, quando este necessita mudar de estado.
- `-(void) revertToPreviousState`: Tem o papel de fazer o *bot* retornar a seu estado anterior.
- `-(void) update`: Esse método é chamado no final do *loop* principal do jogo, onde o método `execute` do estado é chamado, a velocidade vetorial é calculada, e é verificado se o *bot* possui a gema. Em caso positivo, o seu estado muda para TakeGemToBase. Foi feito dessa forma porque não importa em qual estado o *bot* se encontra, se o mesmo tiver posse da gema, deve levá-la à sua base aliada.

A seguir, é descrita a função de cada estado ilustrado na Figura 23.

- **Andando procurando a gema**: No começo do jogo, os *bots* atacantes devem percorrer o mapa de modo a atingirem a base inimiga para roubar a gema. Para tanto, foi escrito uma *plist*, de nome *AttackerRoutes*, que guarda seis caminhos pré-definidos. A Figura 25 mostra o arquivo *plist* aberto no Xcode.

  Cada linha possui um identificador do caminho (coluna "Key") e cada identificador possui um conjunto de caminhos (coluna "Value"). Os números dos conjuntos de caminhos representam os nós do grafo a serem percorridos.

  Quando o *bot* entra no estado "Andando procurando a gema" (remete à classe SearchEnemyGem, Apêndice I), a classe lê o arquivo *plist*, e sorteia os caminhos utilizando uma função randômica da linguagem Objective-C. Em seguida, o jogador automatizado deve seguir a lista de nós de forma sequencial começando do primeiro

Key	Type	Value
▼ Root	Dictionary	(6 items)
1	String	1,20,5,10,12,15,16
2	String	1,20,5,11,15,16
3	String	1,4,18,9,15,16
4	String	1,4,18,9,14,17,16
5	String	1,4,18,3,2,6,7,19,16
6	String	1,20,18,9,14,13,17,16

Figura 25: Arquivo *plist* dos Caminhos Pré-definidos

nó da lista, ou do último, dependendo para qual base ele está indo. O *bot* utiliza o Comportamento de Direção Seguir Caminho, a fim de que seu movimento não pareça muito "robotizado".

Se a gema inimiga estiver no chão, fora de sua base, calcula-se o menor caminho para a gema com base no algoritmo A*, através de uma instância da classe DetectGem.

Se o jogador automatizado detectar um inimigo perto de si, deve mudar o seu estado para "Atacando". Caso ele consiga pegar a gema inimiga, deve mudar o seu estado para "Levando gema inimiga à base aliada". Caso algum aliado roube a gema inimiga e a dificuldade do jogo estiver em "Fácil", o *bot* atacante que está sem a gema deve mudar o seu estado para "Protegendo o portador da gema inimiga".

- **Atacando**: Esse estado remete à classe Attack (Apêndice J), e é responsável - além de ativar o ataque do *bot* - de utilizar do comportamento Seguir para perseguir o inimigo enquanto este está dentro de seu campo de detecção. Caso o inimigo saia do campo de visão do jogador automatizado ou é morto, o estado é alterado para "Melhor caminho à base".

- **Melhor caminho à base**: Esse estado remete à classe BestPathToBase (Apêndice K). Quando o *bot* chama o método *enter* dessa classe, o algoritmo A* é executado a fim de que o menor caminho à base inimiga seja calculado. Em seguida, o Comportamento de Direção Seguir Caminho é utilizado para que o jogador automatizado consiga atingir a base com sucesso.

  Esse estado também cuida de formar o menor caminho até a gema inimiga que se encontra fora da base. As demais condições para mudança de estados são similares ao estado "Andando procurando a gema".

- **Protegendo o portador da gema inimiga**: Representação da classe ProtectTheGemCarrier (Apêndice L), esse estado tem o papel de orientar o jogador automatizado a "escoltar"o *bot* ou jogador humano que está portando a gema inimiga. Essa escolta, se dá pelo comportamento Seguir, onde o alvo é o jogador que porta a gema inimiga.

Se o *bot* encontra algum inimigo em seu caminho, logo muda para o estado "Atacar". Utiliza-se o Comportamento de Direção Seguir para que a escolta seja feita da melhor forma possível.

- **Levando a gema inimiga à base aliada**: Esse estado remete à classe TakeGemToBase (Apêndice M). Assim que o *bot* pega uma gema (aliada ou inimiga), esse estado é alocado (independente de qual estado o jogador automatizado esteja anteriormente). Similar ao estado "Andando procurando a gema", o *bot* sorteia os caminhos pré-definidos à sua base caso ele roube a gema na base inimiga. Se a gema roubada estiver longe de sua base de origem, o algoritmo A* é executado a fim de que o menor caminho à sua base de origem seja calculado.

  O *bot* apenas interrompe o seu trajeto, caso chegue em sua base ou se é morto. Jogadores inimigos são ignorados.

6.3.2 Perfil Defensor

O jogador automatizado defensor é responsável por assegurar que a gema aliada não será roubada. Caso isto aconteça, ele deve seguir o "ladrão" da gema, e atacá-lo até que a gema esteja em seu poder ou de um aliado.

Elaborou-se uma Máquina de Estados para este perfil, onde cada comportamento é um estado diferente da máquina. A Figura 26 indica um diagrama de estados especificado para o perfil atacante de forma a facilitar a compreensão e a modelagem do comportamento do jogador automatizado.

A Figura 27 demonstra o diagrama de classes para o comportamento do jogador automatiado defensor.

Também foi utilizado o Padrão de Projeto *State* para construir o diagrama. A classe BotDefender é bastante similar à BotAttacker, com a diferença de que aquela inicia a máquina de estados com o estado "Patrulhando a base" (Apêndice H).

A seguir, é descrita a função de cada estado ilustrado na Figura 26.

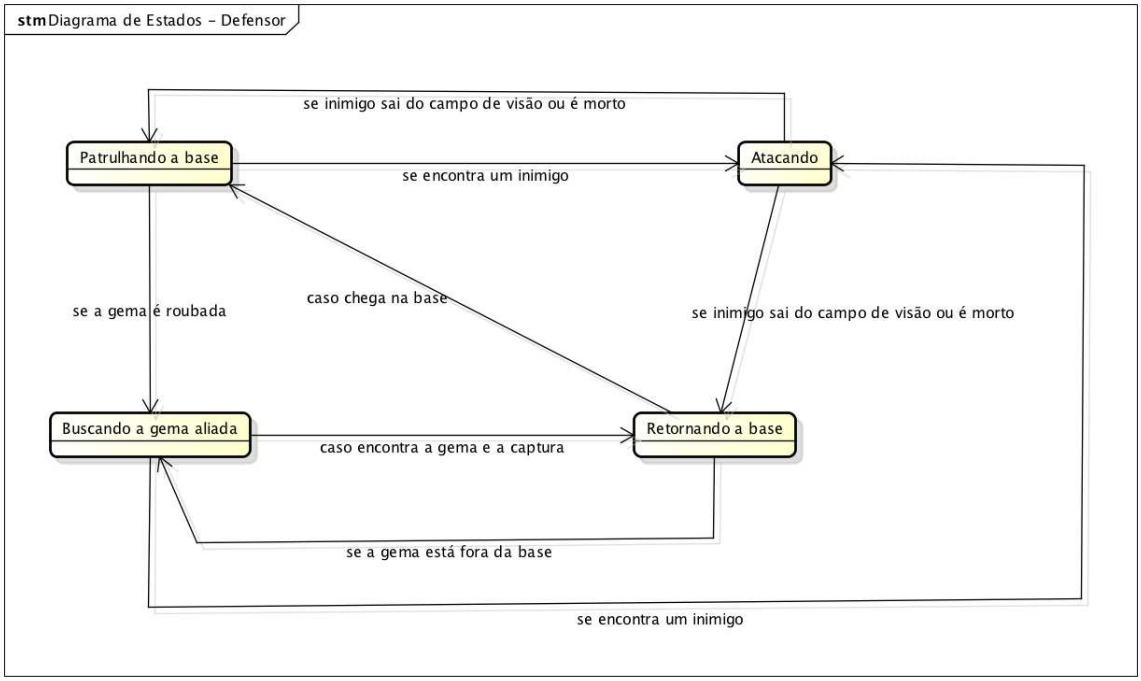

Figura 26: Diagrama de Estados - Perfil Defensor

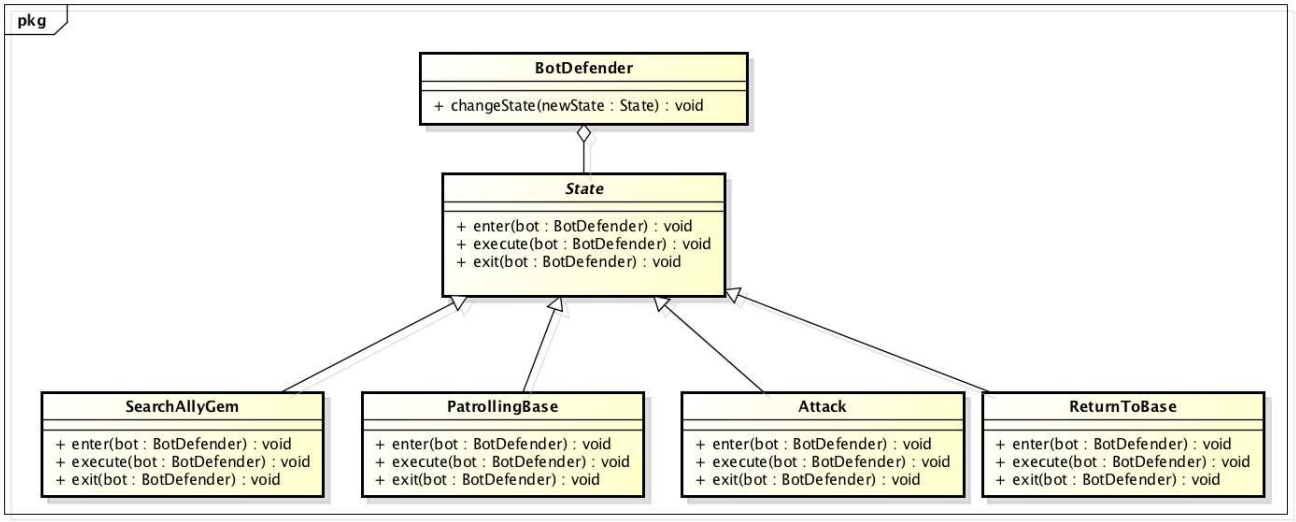

Figura 27: Diagrama de Classes - Perfil Defensor

- **Patrulhando a Base**: Esse estado, representação da classe PatrollingBase (Apêndice N), é responsável por guiar o *bot* a andar sistematicamente em sua base aliada, a fim de proteger a sua gema. O jogador automatizado é orientado a andar por três posições estratégicas em sua base. As Figuras 28 e 29 ilustram as posições que os *bots* defensores de cada time devem atingir.

  Para que isso seja feito, foi escrito um arquivo *plist* que guarda as informações das posições da trajetória do patrulhamento (Figura 30).

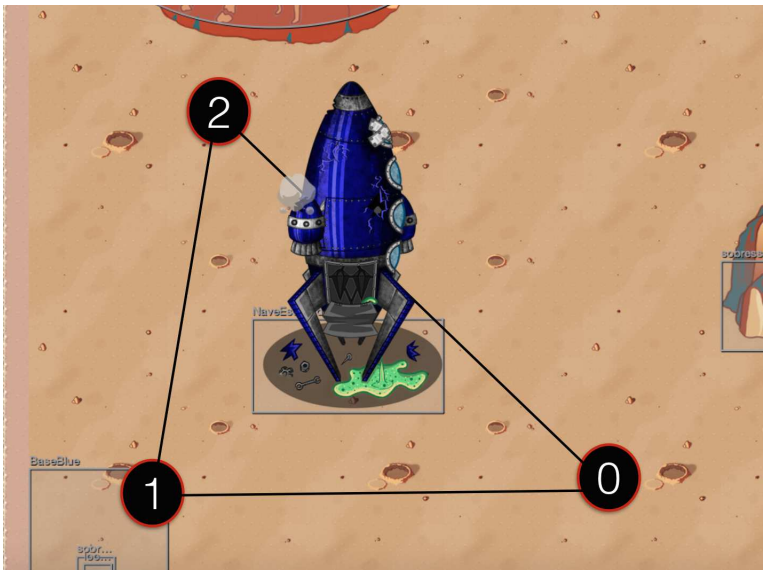

Figura 28: Representação do Caminho na Base Azul

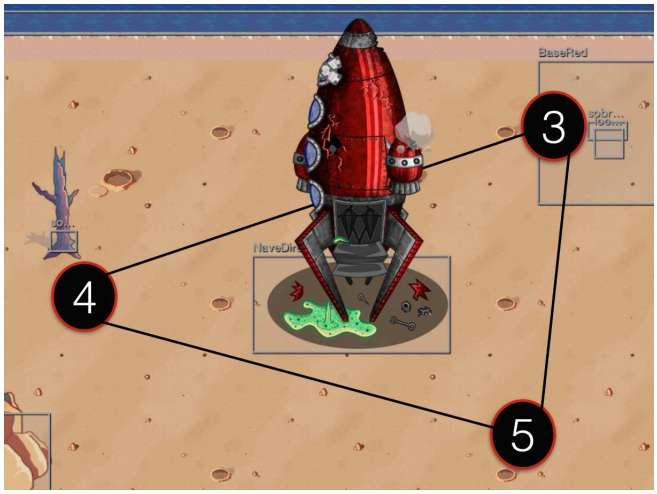

Figura 29: Representação do Caminho na Base Vermelha

A leitura dessa *plist* é feita no método `enter` da classe, e, em seguida, o *bot* segue esses pontos de forma aleatória. Caso um jogador inimigo entre no seu raio de visão, o estado é alterado para "Atacando". Caso a gema aliada seja roubada, o estado é alterado para "Buscando a Gema Aiada".

- **Atacando**: Bastante semelhante ao estado "Atacando" do perfil atacante. Remete à classe AttackDefender (Apêndice O), e a maior diferença para o perfil atacante é que quando o inimigo sai do campo de visão do jogador automatizado, o estado anterior é alocado (linha 25 do Apêndice).

- **Buscando a Gema Aliada**: Quando a gema é roubada, o *bot* defensor deve procurar a gema e o "ladrão". Esse estado tem esse papel (Apêndice P). A cada *loop* do jogo, verifica-se se a gema ainda está em posse do inimigo, e caso afirmativo, o

Key	Type	Value
▼ Root	Dictionary	(6 items)
0	String	135.92,639.13
1	String	134.77,283.07
2	String	728.37,267.35
3	String	1881,1910
4	String	1350,1602
5	String	1911,1572

Figura 30: Arquivo *plist* que Contém a Tragetória dos Defensores

comportamento de direção *Seek* é utilizado, a fim de que o *bot* persiga o ladrão da gema (linha 53).

Caso a gema esteja no chão em algum lugar no mapa, o método *detectBotAndGem* é chamado para que um caminho até a gema seja calculado (linha 36).

A Figura 31 mostra uma tela do jogo onde o *bot* inimigo (com a barra vemelha em cima de si) está seguindo o jogador humano que roubou a gema.

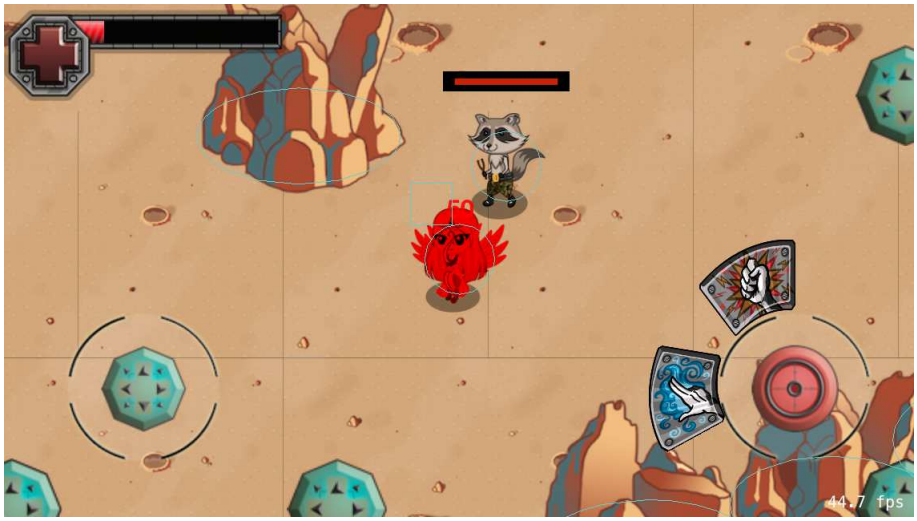

Figura 31: Tela do Jogo

- **Retornando à base**: Caso o *bot* defensor consiga pegar a gema, o estado "Retornando à base"será alocado (Apêndice Q).

  Assim que o método *enter* da classe é chamado, o menor caminho à sua base é calculado, utilizando o algoritmo A*. Para tanto, o Comportamento de Direção Seguir Caminho é utilizado. Assim que o jogador automatizado consiga se aproximar o suficiente de sua base (linha 49 do código), a gema é devolvida e o estado "Patrulhando a Base" é alocado.

6.3.3 Perfil Seguidor

Quando o jogo está no nível Fácil, definiu-se um jogador automatizado que seguisse o jogador humano durante toda a partida. Ele estaria encarregado de "protegê-lo"em todo o seu trajeto, a fim de que o jogo se torne mais fácil. A Figura 32 ilustra a máquina de estados para esse perfil.

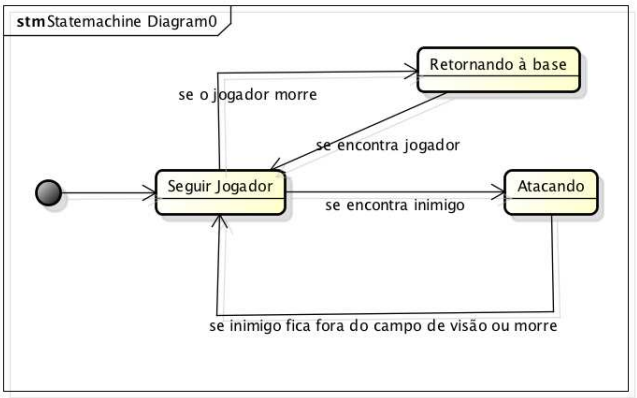

Figura 32: Diagrama de Estados - Perfil Seguidor

O diagrama é bastante simples, com apenas três estados. O estado "Seguir jogador", a cada *loop* do jogo, verifica se o jogador humano se encontra muito perto do jogador automatizado. Em caso positivo, ele apenas utiliza o Comportamento Direcional Seguir para se aproximar do jogador humano. Caso contrário, calcula-se a menor distância entre o *bot* e o jogador através do algoritmo A*, e o Comportamento de Direção Seguir Caminho é acionado.

A Figura 33 mostra uma tela do jogo em execução, quando o jogador humano encontra inimigos. Os *bots* são aqueles que possuem barras coloridas em cima de seus personagens. A barra verde indica um *bot* alido ao jogador, e a barra vermelha indica um jogador inimigo.

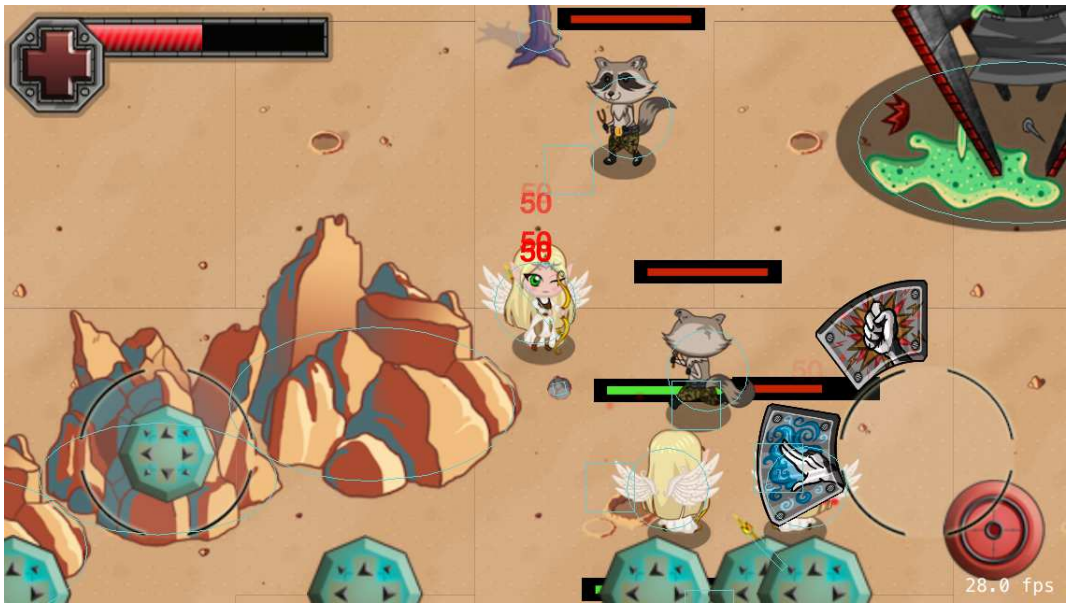

Figura 33: Tela do Jogo em Execução

6.3.4 Nível de Dificuldade

Os jogadores automatizados são capazes de enfrentar outros *bots* semelhantes e também jogadores humanos. Para tanto, definiu-se um balanceamento da dificuldade dos *bots* de forma que o usuário não se sinta frustrado com um nível de dificuldade muito alto ou entediado por causa do jogo ser muito fácil. Nesse contexto, foram estabelecidos três níveis de dificuldade que podem ser escolhidos pelo usuário: Fácil, Médio e Difícil.

A seguir, é descrito em detalhes como cada nível lida com a dificuldade do jogo.

- Fácil: Possui as seguintes particulariedades:
  - Os *bots* demoram para atirar em um inimigo. O método *applyErrorAccordingToDifficulty*, como poder ser visto no Apêndice A, linha 104, é executado a cada *loop* do jogo, e a probabilidade de atirar em um inimigo é de apenas 5% dos números sorteados. Quando o *bot* atira, a probabilidade de acertar o tiro é de 50%. Todos valores de porcentagem de erro dos tiros, em todos os níveis de dificuldade, foram definidos após vários testes e ajustes, até que se chegasse em um número "ideal".
  - No início do jogo, o *bot* atacante aliado do jogador humano não demora para chegar na base inimiga, ele se dirige à base sem espera alguma (Apêndice I, linha 114). Isso pode ajudar o jogador iniciante que naturalmente demora à achar a base inimiga, porque quando o mesmo consegue chegar na base, os *bots* aliados já estão atacando os jogadores inimigos.

  - O jogador automatizado inimigo do jogador humano demora para atravessar o mapa. O método *changeCurrentPath* é chamado, e a tragetória do *bot* é alterada, fazendo ele voltar alguns nós depois que atinge uma certa distância.
  - O perfil Seguidor é alocado em apenas um *bot* aliado do jogador humano.

- Médio: Possui as seguintes particulariedades:
  - Os *bots* demoram menos para atirar em um inimigo. A probabilidade de atirar em um inimigo, quando o encontra, é de 40%. Quando o jogador automatizado atira, a probabilidade de acertar o tiro é de 50%.
  - No início do jogo, o *bot* atacante aliado do jogador humano não demora para chegar na base inimiga, ele se dirige à base sem espera alguma.
  - O jogador automatizado inimigo do jogador humano demora para atravessar o mapa.

- Difícil: Possui as seguintes particulariedades:
  - Os *bots* não demoram para atirar em um inimigo. A probabilidade de atirar em um inimigo, quando o encontra, é de 90%. Quando o jogador automatizado atira, a probabilidade de acertar o tiro é de 50%.
  - No início do jogo, o *bot* atacante aliado do jogador humano demora a chegar na base inimiga (Apêndice I, linha 108). Isso aumenta a dificuldade do jogo, pois o jogador humano experiente irá chegar na base inimiga sozinho, enfrentando os outros *bots* sem ajuda.
  - O jogador automatizado inimigo do jogador humano não demora para atravessar o mapa. Isso contribui para a dificuldade, pois os inimigos chegam mais rápido na base para roubar a gema, aumentando a dificuldade para defender a base.
  - Os *bots* defensores atacam somente o ladrão da gema. Eles não se distraem com outros jogadores inimigos. O jogador humano que roubar a gema deve ser mais ágil em fugir, pois os seus inimigos não irão atacar outros até que consigam recuperar a gema.

6.4 Considerações Finais

Foi apresentada neste capítulo uma visão geral do comportamento dos *bots*. O Grafo implementado foi utilizado em todos os perfis de jogador, assim como os Comportamentos de Direção. Todas as funcionalidades e comportamentos foram sendo testadas pelo desenvolvedor à medida que foram sendo concluídas. O nível de dificuldade, em especial, foi realizado seguindo essa metodologia.

Observou-se alguns critérios para aceitação das Histórias de Usuário. São exemplos: fluidez da movimentação dos *bots*, nível de dificuldade adequado, nível de entretenimento considerável, entre outros. Após os testes feitos pelo desenvolvedor, outros tipos de testes foram conduzidos, a fim de aumentar a qualidade do código da inteligência dos jogadores automatizados. O capítulo Resultados Obtidos descreve como esses testes foram conduzidos.

7 Resultados Obtidos

Após a implementação do código, com todas as Histórias de Usuário implementadas, foi necessário avaliar a qualidade do código e do comportamento dos jogadores automatizados, a fim de que os objetivos fosse atendidos de forma plena. Para isso, foram definidos quatro tipos de testes e métricas: testes dos *bots* contra *bots*, complexidade ciclomática do código, testes unitários e testes com usuários.

7.1 Testes dos Bots contra Bots

A fim de se avaliar o comportamento, foram realizados testes com os jogadores automatizados implementados, onde foi analizado o comportamento dos mesmos quando estão atuando na partida sem a intervenção humana, ou seja, um time de *bots* com outro. Para tanto, foram testados os três nívels de dificuldade implementados, durante 10 minutos cada.

No nível fácil e médio, após quatro rodadas de testes, onde os *bots* se enfrentavam sem interferência humana, não houve vencedores, ou seja, os *bots* atacante dos dois times não conseguiram roubar a gema inimiga e levá-la à base aliada.

No nível difícil, de quatro partidas testadas, duas delas houveram vencedores e em três ocorrou empate. Pode-se perceber nos resultados obtidos que, na maioria dos casos, as habilidades dos *bots* que se enfrentam são iguais ou extremameente parecidas, ocasionando um equilibrio perfeito entre os times. Esse resultado não é de todo negativo, pois isso dá espaço para o jogador humano desequilibrar a partida de acordo com a sua habilidade no jogo, aumentando a competitividade e entretenimento da partida.

7.2 Complexidade Ciclomática

Após as Histórias de Usuário serem implementadas, verificou-se a complexidade ciclomática do código utilizando-se a ferramenta XClafiry. A Figura 34 mostra uma análise preliminar feita pela ferramenta.

Como ilustrado na figura, a ferramenta dá a opção de verificar a complexidade ciclomática por método, e filtrá-los de acordo com suas respectivas complexidades. De acordo com McCabe (1976), valores maiores que dez são indicadores que os métodos estão muito complexos e necessitam refatoração, por isso optou-se por filtrar os métodos que possuam complexidade ciclomática acima de dez.

As cinco primeiras classes que aparecem nos resultados (GameLayer, HudLayer,

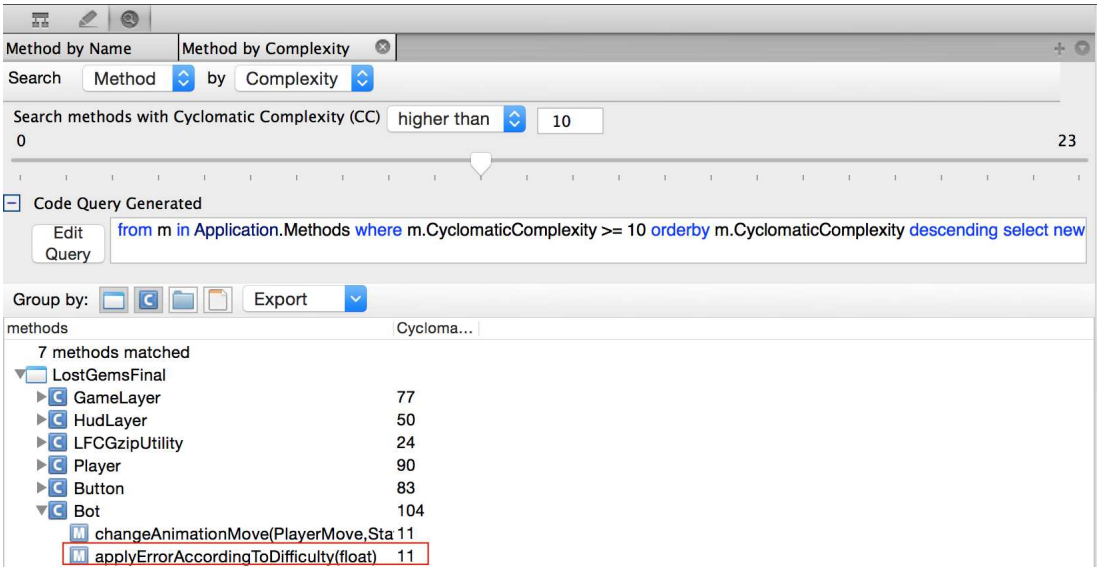

Figura 34: Complexidade Ciclomática Preliminar

LFCGzipUtility, Player e Button) são classes referentes a domínios diferentes do contexto desse trabalho. Portanto, foram ignorados na análise. O método *changeAnimationMove* da classe Bot é um método que trata da animação do movimento dos *bots* e foge do escopo da inteligência do qual este trabalho trata. Portanto, esse método também foi ignorado.

O método *applyErrorAccordingToDifficulty* (marcado em vermelho na Figura 34) é responsável por aplicar o erro e *delay* necessários à bala de acordo com a dificuldade do jogo. Percebeu-se que esse era o único método relacionado à inteligência dos *bots* que tinha complexidade ciclomática acima de dez. Visando diminuir esse valor, realizou-se um trabalho de refatoração do método *applyErrorAccordingToDifficulty*. Após as devidas correções, analisou-se novamente, através do XClarify, a complexidade ciclomática do código. A Figura 35 ilustra os resultados obtidos.

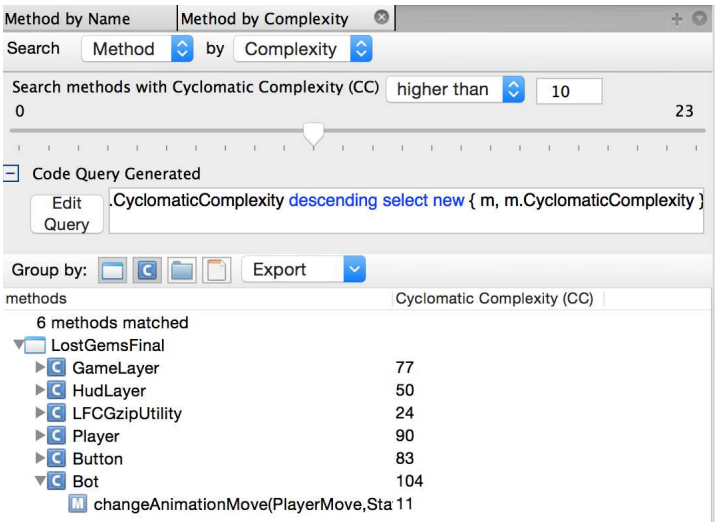

Figura 35: Complexidade Ciclomática Preliminar

Percebeu-se que a complexidade ciclomática do método mudou para abaixo de dez, atendendo o padrão de método simples e fácil de ser testado, de acordo com McCabe (1976).

7.3 Testes Unitários e Cobertura de Código

O teste unitário tem por objetivo testar a menor parte testável do sistema (unidade), geralmente um método. Idealmente, o teste unitário é independente de outros testes, analisando assim cada parte ou funcionalidade individualmente. Nos casos em que existe dependência de métodos ou classes que não estão sob teste, são usados *mocks* (objetos criados para simular o comportamento de objetos reais nos testes) para simular estes objetos ou métodos que são usados pela unidade, desacoplando a unidade sob teste dos componentes dos quais ela é dependente (LUZZI, 2014).

Para o código que implementa o comportamento dos *bots*, foram escritas várias classes de testes que possuem o papel de testar a grande maioria das funcionalidades. Como exemplo de classe de teste, o apêndice S contém a classe TestAttack que testa as funcionalidades da classe `Attack.m`. Definiu-se como objetivo, cobrir no mínimo 90% do código com testes unitários.

A medida que os testes foram sendo escritos, estes eram compilados e executados a fim de se atestar se os testes não falhavam. A Figura 36 é uma tela do Xcode que mostra que todos os testes escritos estavam funcionando e passavam perfeitamente (o marcador verde indica isso).

Também foi analisada a medida de cobertura de código dos testes. O apêndice U mostra a medida da cobertura de código das principais classes que lidam com o comportamento dos *bots*.

As classes que lidam diretamente com o comportamento dos jogadores automatizados estão marcadas em vermelho. A barra azul em frente do nome de cada classe indica a porcentagem da cobertura de código de forma visual. Todas as classes estão ordenadas pela porcentagem de cobertura de forma crescente.

O número da porcentagem pode ser obtido assim que se passa o cursor do *mouse* em cima de uma barra azul. Posicionando o *mouse* em cima da classe SteeringBehaviours, percebe-se que a porcentagem da cobertura da classe é de 91%. Logo, conclui-se que todas as outras classes acima desta possuem porcentagem de cobertura maior ou igual a 91%. Esta análise supre as expectativas de possuir uma cobertura de código acima de 90%.

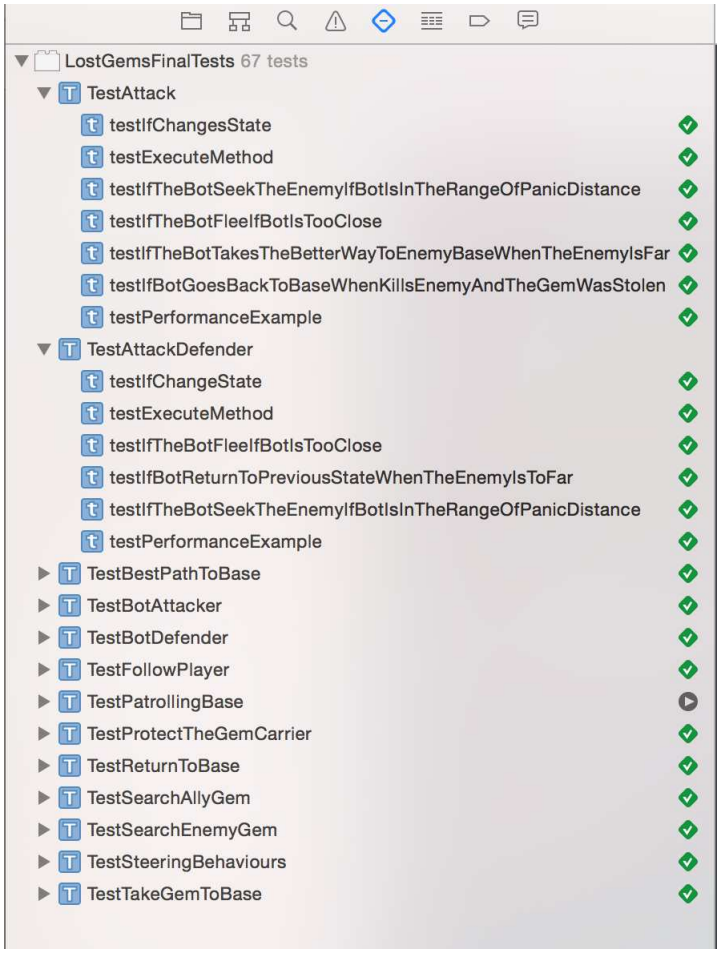

Figura 36: Testes Executados com Sucesso

7.4 Testes com Usuários

Para que a inteligência, dificuldade e nível de diversão dos *bots* fosse avaliada, testes com usuários foram guiados após a implementação das Histórias de Usuário. A estratégia se dividiu nos seguintes passos:

- Foi solicitado à 15 pessoas, na faixa dos 18 aos 25 anos, a jogarem o jogo *Lost Gems* como os *bots* implementados. Todos os usuários que testaram o jogo jogaram no nível fácil, com cinco *bots* aliados e cinco inimigos. O usuário podia lançar habilidades especiais por um curto espaço de tempo, tais como cura instântanea de si mesmo, e tiros mais potentes que os demais jogadores.

- Após os usuários jogarem o jogo por um certo tempo, foi aplicado um questionário de avaliação das impressões do usuário acerca dos *bots*. Pode-se observar o questionário no apêndice U. Na elaboração das questões, priorizou-se perguntas objetivas, que coletassem as principais impressões dos jogadores em relação ao comportamento dos jogadores automatizados.

- Os resultados das questões objetivas podem ser vistos nas Figuras 37, 38 e 39.

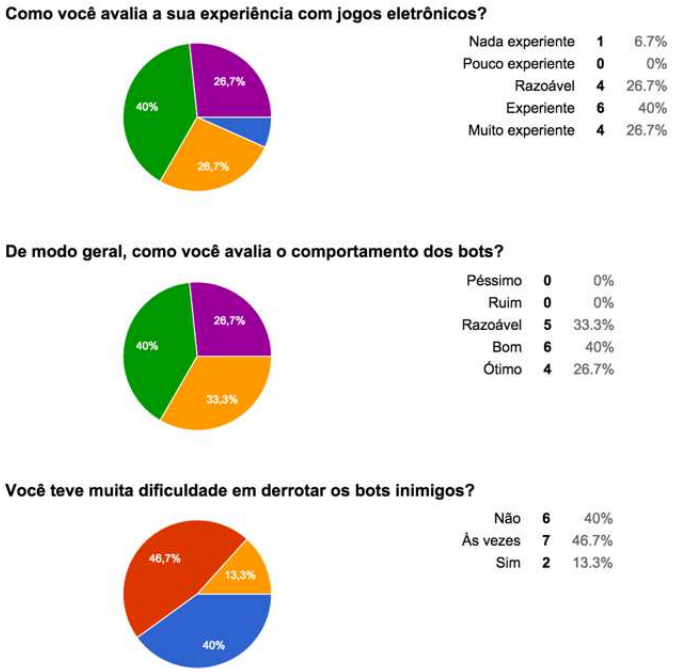

Figura 37: Estatísticas dos Resultados (Parte I)

Empregou-se uma análise quantitativa e qualitativa dos resultados. Percebeu-se que a maioria dos usuários já possuíam experiência prévia com jogos eletrônicos (o que influencia diretamente na pesquisa). Um valor consideravel de usuários tiverem algum tipo de dificuldade em enfrentar os *bots* inimigos, e 53,3% destes acharam o jogo razoável, ou difícil ou muito difícil. Grande parte dos usuários (33,3%) se sentiram sozinhos no jogo em relação aos *bots* aliados.

Na última questão, percebeu-se que 60% dos usuários conseguiram identificar facilmente que não estavam jogando com jogadores humanos, e sim com jogadores automatizados. Percebeu-se, então, que a proposta do objetivo geral deste trabalho não se adequou ao resultado obtido. Trabalhar em cima da inteligência dos *bots* para que estes se comportassem como jogadores humanos demandaria uma carga de trabalho muito além do esperado, explorando, muito provavelmente, algoritmos de *learning* (LANGLEY, 2011), *reasoning* (WOS et al., 1984), redes neurais (BISHOP, 1995), bases de conhecimento especializadas (PARSAYE; CHIGNELL; KHOSHAFIAN, 1989), entre outras técnicas avançadas de Inteligência Artificial. Porém, a questão de pesquisa foi atendida no que se refere ao equilibrio das equipes envolvidas em uma partida, mantendo a competitividade do jogo.

As respostas subjetivas, onde se empregou uma abordagem qualitativa para a análise, se resumiram, em sua maioria, em:

- Os *bots* aliados não são vistos frequentemente, e

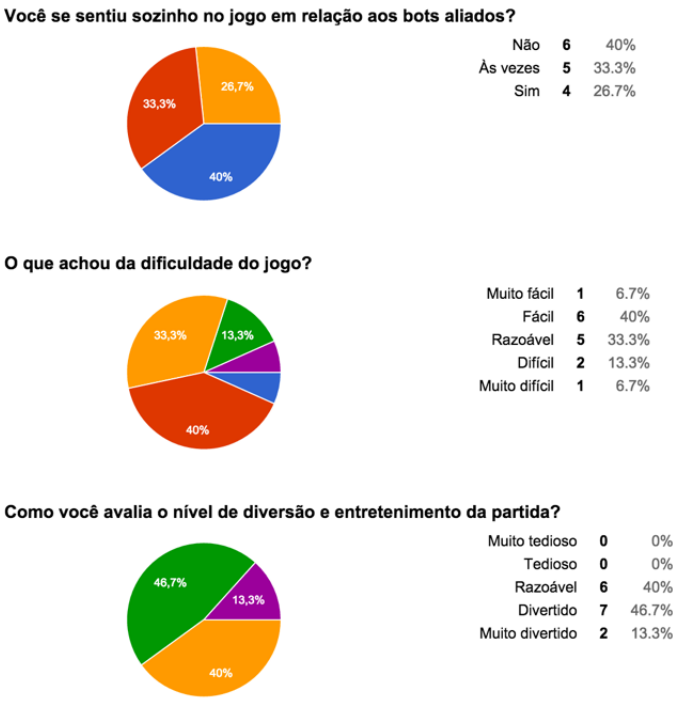

Figura 38: Estatísticas dos Resultados (Parte II)

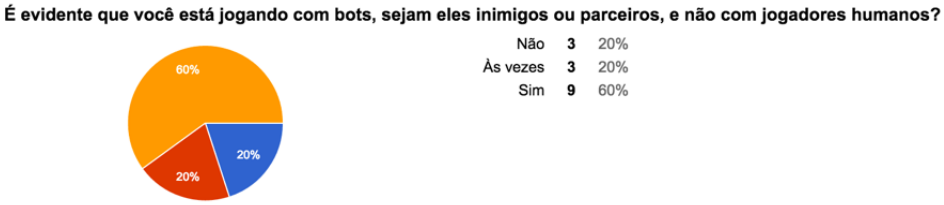

Figura 39: Estatísticas dos Resultados (Parte III)

- A dificuldade ficou adequada para alguns.

Tendo em vista esses *feedbacks*, e procurando utilizar a modalidade de pesquisa-ação, pensou-se em criar o, até então inexistente, Perfil Seguidor, a fim de que os novos jogadores não se sentissem sozinhos no meio do jogo. Procurou-se aumentar ainda a velocidade máxima de movimentação dos jogadores automatizados, adequando-a à velocidade máxima do jogador humano, para que os jogadores automatizados e os humanos andassem juntos, sem diferença de velocidade.

7.5 Consideração Finais

Neste capítulo, procurou-se descrever as principais técnicas de testes e análises utilizadas, bem como os seus resultados. Tais resultados se tornaram importantíssimos

para a correção do comportamento dos jogadores automatizados, a fim de que o objetivo principal e os objetivos específicos fossem atendidos de forma satisfatória.

As modalidades de pesquisa quantitativa, qualitativa e pesquisa-ação se mostraram essenciais para que essas correções fossem conduzidas. Com dados concretos, obteve-se maior propriedade e credibilidade para se modificar alguns comportamentos e melhorar a qualidade do código escrito.

8 Conclusão

O objetivo geral deste trabalho (*Desenvolver jogadores automatizados no Lost Gems, utilizando-se de Máquina de Estados, Comportamentos de Direção, Grafos e Algoritmos de Busca, visando simular o comportamento de jogadores humanos e equilibrar as equipes envolvidas em uma partida, mantendo a competitividade do jogo*) foi atendido em parte. Os algoritmos de Máquina de Estados, Comportamentos de Direção, Grafos e Algoritmos de Busca, foram implementados conforme especificado nos capítulos anteriores, e orientando-se pelas boas práticas estudadas nas áreas de Engenharia de Software e Inteligência Artificial. Inclusive, novas funcionalidades envolvendo esses algoritmos foram implementadas com base no resultado dos testes conduzidos.

Em relação ao objetivo "simular o comportamento de jogadores humanos", o resultado dos testes com usuários mostraram que essa meta não foi completamete atendida. Para que fosse atingido o objetivo, seria necessário a aplicação de técnicas de Inteligência Artificial mais avançadas, o que foge do escopo desse trabalho.

Verificou-se que a parte do objetivo "equilibrar as equipes envolvidas em uma partida, mantendo a competitividade do jogo" foi atingido, pois grande parte dos usuários revelou que achou a dificuldade adequada, e o jogo divertido.

Todos os objetivos específicos foram atingidos, ou seja, as máquinas de estados para os diferentes perfis de *bots* foram implementadas; diferentes estratégias de grafos foram estudadas; ações específicas dos jogadores automatizados foram desenvoldidas (andar pelo mapa, desviar de obstáculos, capturar a gema inimiga, entre outros); dados de impressões de usuários acerca do jogo foram coletados, e as boas práticas de Engenharia de Software foram aplicadas (tais como metodologia ágil, testes unitátios e análise estática do código).

A classe que obteve o menor valor de cobertura de código de testes unitários teve a porcentagem de 91%, atingindo a meta pré-estabelecida de uma cobertura de código acima de 90%. No final do desenvolvimento e das devidas correções, nenhum método obteve número de complexidade ciclomática acima de dez, conforme recomendado pelo autor McCabe (1976).

8.1 Direitos Autorais

Após o final de todo esse ciclo inicial de desenvolvimento, foi pensado na questão de direitos autorais do código escrito para os *bots* (já que o jogo como um todo foi desenvolvido por cinco desenvolvedores). Para se evitar problemas legais futuros, um documento de reconhecimento da implementação dos jogadores automatizados foi escrito

e assinado por cada um dos desenvolvedores do jogo *Lost Gems* (apêndice W), conferindo ao autor desse TCC, os créditos por ter implementado as estratégicas de Inteligência Artificial, agregando ao jogo uma visão preliminar nesse contexto.

8.2 Trabalhos Futuros

Pretende-se, futuramente, melhorar a inteligência dos *bots* a fim de que o jogo se torne ainda mais divertido e desafiador. O objetivo principal do desenvolvimento dos jogadores automatizados é anexar essa funcionalidade à versão final do jogo, e finalmente publicá-lo na loja da *Apple* chamada *App Store*, onde ficará disponível para *download* em dispositivos iOS (*iPhone*, *iPad* e *iPod*).

O desenvolvimento da Inteligência Artificial dos jogadores automatizados em *Lost Gems* não foi uma tarefa simples e rápida. As estratégias clássicas já estabelecidas e amplamente difundidas no meio científico apoiaram fortemente esse desenvolvimento, permitindo estabelecer algoritmos elegantes e funcionais de IA para jogadores autônomos.

Referências

APPLE. *About Sprite Kit*. 2014. Disponível em: <https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html>. Citado 2 vezes nas páginas 45 e 46.

BEVILACQUA, F. *Understanding Steering Behaviors: Seek*. 2012. Disponível em: <http://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-seek--gamedev-849>. Citado 3 vezes nas páginas 32, 33 e 34.

BISHOP, C. M. *Neural networks for pattern recognition*. [S.l.]: Oxford university press, 1995. Citado na página 83.

BUCKLAND, M. *Programming Game AI by Example*. 1. ed. [S.l.]: Wordware Publishing, 2005. 495 p. Citado 5 vezes nas páginas 13, 27, 34, 35 e 36.

CORMEM, T. H. et al. *Algoritmos: teoria e prática*. 2. ed. Rio de Janeiro: Campus, 2002. Citado 3 vezes nas páginas 13, 37 e 38.

DENCKER, A. d. F. M. *Métodos e técnicas de pesquisa em turismo*. 4. ed. São Paulo: Futura, 2001. Citado na página 49.

FEOFILOFF, P. et al. *Uma introdução sucinta à teoria dos grafos*. 2011. Disponível em: <http://www.ime.usp.br/~pf/teoriadosgrafos/>. Citado 2 vezes nas páginas 13 e 37.

FERRAZ, R. M. *Conceitos de Programação: Complexidade Ciclomática*. 2008. <http://logbr.reflectivesurface.com/2008/11/12/conceitos-de-programacao-complexidade-ciclomatica/>. [Online; accessed 12-november-2015]. Citado na página 40.

GAMMA, E. et al. *Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos*. Porto Alegre: Bookman, 2000. 364 p. Citado 3 vezes nas páginas 13, 40 e 41.

GONçALVES, C. A. et al. *Projetos e relatórios de pesquisa em Administração*. São Paulo: Atlas, 2004. Citado na página 49.

HASTJARJANTO, T.; JEURING, J.; LEATHER, S. A dsl for describing the artificial intelligence in real-time video games. *3rd International Workshop on*, p. 18, May 2013. Citado na página 26.

HENDERSON-SELLERS, B.; TEGARDEN, D. The application of cyclomatic complexity to multiple entry/exit modules. *Center for Information Technology Research Report*, n. 60, 1993. Citado na página 40.

HOSEA, S.; HARIKRISHNAN, V.; RAJKUMAR, K. Artificial intelligence. In: *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*. [S.l.: s.n.], 2011. v. 4, p. 124–129. Citado na página 29.

HU, W.; ZHANG, Q.; MAO, Y. Component-based hierarchical state machine - a reusable and flexible game ai technology. In: *Information Technology and Artificial Intelligence Conference (ITAIC), 2011 6th IEEE Joint International*. [S.l.: s.n.], 2011. v. 2, p. 319–324. Citado na página 30.

JR., J. H. et al. *Fundamentos de métodos de pesquisa em Administração*. Porto Alegre: Bookman, 2005. Citado na página 49.

KEITH, C. *Agile Game Development with Scrum*. 1. ed. [S.l.]: Addison-Wesley Professional, 2010. Citado na página 53.

LAKATOS, E. M.; MARCONI, M. d. A. *Fundamentos de metodologia científica*. 4. ed. São Paulo: Atlas, 2001. Citado na página 49.

LANGLEY, P. The changing science of machine learning. *Machine Learning*, Springer US, v. 82, n. 3, p. 275–279, 2011. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1007/s10994-011-5242-y>. Citado na página 83.

LI, H. Application of artificial intelligence in computer aided instruction. In: *Test and Measurement, 2009. ICTM '09. International Conference on*. [S.l.: s.n.], 2009. v. 2, p. 221–224. Citado na página 29.

LOPES, S. A. *Métodos Finitos em Matemática*. [S.l.]: Departamento de Matemática da Faculdade de Ciências da Universidade do Porto, 2009. Citado na página 37.

LUZZI, L. *Testes unitários e TDD – Conceitos Básicos*. 2014. Disponível em: <http://www.mobiltec.com.br/blog/index.php/testes-unitarios-e-tdd-conceitos-basicos/>. Citado na página 81.

MCCABE, T. A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2, n. 4, p. 308–320, Dec 1976. ISSN 0098-5589. Citado 5 vezes nas páginas 39, 40, 79, 81 e 87.

NING, S.; YAN, M. Discussion on research and development of artificial intelligence. In: *Advanced Management Science (ICAMS), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010. v. 1, p. 110–112. Citado na página 29.

PARSAYE, K.; CHIGNELL, M.; KHOSHAFIAN, S. Intelligent databases. object-oriented, deductive hypermedia technologies. *New York: Wiley, 1989*, v. 1, 1989. Citado na página 83.

RABIN, S. *AI game programing wisdow*. Massachusets: Charles River Media, 2002. Citado 3 vezes nas páginas 38, 39 e 57.

RECIO, G. et al. Antbot: Ant colonies for video games. *Computational Intelligence and AI in Games, IEEE Transactions on*, v. 4, n. 4, p. 295–308, Dec 2012. Citado 2 vezes nas páginas 30 e 31.

REYNOLDS, C. W. Steering behaviors for autonomous characters. In: *Game developers conference*. [S.l.: s.n.], 1999. v. 1999, p. 763–782. Citado 5 vezes nas páginas 31, 32, 33, 34 e 58.

SALES, S. Padrão de projeto: State. *Faculdade de Ciências e Tecnologia – FIB - Centro Universitário da Bahia*, p. 4, Setembro 2008. Citado 2 vezes nas páginas 40 e 41.

SAMARA, B. S.; BARROS, J. C. *Pesquisa de Marketing: Conceitos e metodologia*. 4. ed. São Paulo: Pearson Prentice Hall, 2007. Citado na página 49.

SCHETINGER, V. et al. User stories as actives for game development. *SBC - Proceedings of SBGames 2011*, Nov 2011. Citado 2 vezes nas páginas 53 e 54.

SCHWABER, K.; SUTHERLAND, J. *Guia do Scrum: Um guia definitivo para o Scrum: As regras do jogo.[Sl]*. 2013. Citado na página 42.

SHIFFMAN, D. et al. *The nature of code*. [S.l.]: D. Shiffman, 2012. Citado 2 vezes nas páginas 31 e 32.

SILVA, J. B. D. *Estudo comparativo entre algoritmo A* e busca em largura para planejamento de caminho de personagens em jogos do tipo Pacman*. [S.l.]: Universidade Regional de Blumenal, 2005. Citado 2 vezes nas páginas 37 e 38.

TIWARI, U.; KUMAR, S. Cyclomatic complexity metric for component based software. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 39, n. 1, p. 1–6, fev. 2014. ISSN 0163-5948. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/2557833.2557853>. Citado 2 vezes nas páginas 39 e 40.

TRIPP, D. Pesquisa-ação: uma introdução metodológica. *Educação e pesquisa*, SciELO Brasil, v. 31, n. 3, p. 443–466, 2005. Citado 2 vezes nas páginas 50 e 51.

VERGARA, S. C. *Projetos e relatórios de pesquisa em administração*. 3. ed. São Paulo: Atlas, 2000. Citado na página 49.

WAINER, J. Métodos de pesquisa quantitativa e qualitativa para a ciência da computação. *PUC - Rio*, 2007. Citado 2 vezes nas páginas 49 e 50.

WOS, L. et al. Automated reasoning: introduction and applications. Prentice Hall Inc., Old Tappan, NJ, 1984. Citado na página 83.

Apêndices

APÊNDICE A – Código Fonte da classe Bot

```
1 //
2 //  Bot.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 08/09/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "Bot.h"
10 #import "SKTUtils.h"
11 #import "SteeringBehaviors.h"
12 #import "RetangleCollision.h"
13 #import "DetectEnemy.h"
14 #import "LifeBar.h"
15 #import "Elf.h"
16 #import "Raccoon.h"
17 #import "SnowMan.h"
18 #import "Graph.h"
19 #import "ListVertex.h"
20 #import "PathManager.h"
21 #import "LifeBar.h"
22 #import "Gem.h"
23 #import "DetectGem.h"
24 #import "FollowPlayer.h"
25
26 #define MASS 1
27 #define kBotHeight 42
28 #define RETANGLE_COLLISION_WIDTH 50
29 #define RETANGLE_COLLISION_HEIGHT 50
30
31 @interface Bot ()
32
33 @property (nonatomic) LifeBar *llifeBar;
34 @property (nonatomic) float actualTimestamp;
35 @property (nonatomic, strong) NSMutableArray *currentPath;
36
37 @end
38
39 @implementation Bot
40
41 -(void)configureIntelligence {
42
43     self.retangle = [RetangleCollision spriteNodeWithColor:[UIColor blackColor] size:
              CGSizeMake(RETANGLE_COLLISION_WIDTH, RETANGLE_COLLISION_HEIGHT)];
44     self.retangle.bot = self;
45     self.previousTimestamp = 0;
46     self.detectEnemy = [[DetectEnemy alloc] init];
47     self.detectGem = [[DetectGem alloc] init];
48     self.detectEnemy.actualBot = self;
49     self.steeringBehaviors = [[SteeringBehaviors alloc] init];
50     self.steeringBehaviors.player = self.player;
```

```
51    self.maxSpeed = MAX_SPEED;
52    self.mass = MASS;
53    self.isAttacking = NO;
54    self.displayLink = [CADisplayLink displayLinkWithTarget:self selector:@selector(update)
              ];
55    self.pathManager = [[PathManager alloc] init];
56    self.pathManager.bot = self;
57    self.isInView = NO;
58    [self.retangle configureCollisionofRetanngle:self.retangle];
59    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
60        [self addLifeBar];
61    });
62    [self addChild:self.retangle];
63    [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop] forMode:NSRunLoopCommonModes];
64
65    if (!self.isBotFollower) {
66        [self initActions];
67    }
68    else {
69        [self isFollowerAccordingToDifficulty];
70    }
71
72}
73
74-(void) initActions {
75
76}
77
78-(void) changeEstate:(id<StateBot>)newState {
79
80}
81
82-(void) revertToPreviousState {
83
84}
85
86
87-(void) isFollowerAccordingToDifficulty {
88    self.currentState = [[FollowPlayer alloc] init];
89    [self.currentState enter:self];
90}
91
92-(void) applySteeringForce:(CGPoint) steeringForce {
93    steeringForce = CGPointMultiplyScalar(steeringForce, 500 * RandomFloatRange(0.7, 1));
94    CGVector impulseVector = CGVectorMake(steeringForce.x, steeringForce.y);
95    [self.physicsBody applyImpulse:impulseVector];
96}
97
98
99
100-(float) applyErrorAccordingToDifficulty:(float)angle {
101    float randomNumber = RandomFloatRange(0, 100);
102
103    switch (self.difficulty) {
104        case kEasy:
105            if (randomNumber > 5) {
106
```

```
107                 return INT_MAX;
108             }
109             else {
110                 if ((int)randomNumber%2 == 1) {
111                     return 0.5;
112                 }
113             }
114             break;
115         case kMedium:
116             if (randomNumber > 60) {
117 
118                 if ((int)randomNumber%2 == 0) {
119                     return 0.5;
120                 }
121                 else{
122                     return -0.5;
123                 }
124             }
125             break;
126         case kHard:
127             if (randomNumber > 90) {
128                 if ((int)randomNumber%2 == 0) {
129                     return 0.5;
130                 }
131                 else{
132                     return -0.5;
133                 }
134             }
135             break;
136 
137         default:
138             break;
139     }
140     return 0;
141 }
142 
143 -(int) closerNode {
144 
145     float x = 999999;
146     int nodeID = 0;
147 
148     for (ListVertex *node in self.graph.nodesList) {
149 
150         if (CGPointDistance(node.position, self.position) < x) {
151             nodeID = node.iD;
152             x = CGPointDistance(node.position, self.position);
153         }
154     }
155     return nodeID;
156 }
157 
158 @end
```

APÊNDICE B – Código da classe SteeringBehaviours

```
1//
2//  SteeringBehaviors.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 12/06/15.
6//  Copyright (c) 2015 Bruno Rodrigues de Andrade. All rights reserved.
7//
8
9#import "SteeringBehaviors.h"
10#import "SKTUtils/SKTUtils.h"
11#import "Bot.h"
12
13#define MAX_SEE_AHEAD 50
14#define MAX_DISTANCE_TO_NODE 150
15
16@implementation SteeringBehaviors
17
18- (instancetype)init
19{
20    self = [super init];
21    if (self) {
22        self.iterator = 0;
23    }
24    return self;
25}
26
27-(CGPoint) seek:(CGPoint)targetPos andBot:(Bot *)bot{
28
29    CGPoint desiredVelocity = CGPointMultiplyScalar( CGPointNormalize(CGPointSubtract(
              targetPos, bot.position)), MAX_SPEED);
30
31    return CGPointSubtract(desiredVelocity, bot.velocityPointsPerFrame);
32
33}
34
35-(CGPoint) flee:(CGPoint)targetPos andBot:(Bot *)bot{
36    double panicDistance = 100;
37
38    if(CGPointDistance(bot.position, targetPos) > panicDistance){
39        return CGPointMake(0, 0);
40    }
41
42    CGPoint desiredVelocity = CGPointMultiplyScalar(CGPointNormalize(CGPointSubtract(bot.
              position, targetPos)),MAX_SPEED) ;
43
44    return CGPointSubtract(desiredVelocity, bot.velocityPointsPerFrame);
45}
46
```

```
47-(CGPoint) followPath:(NSMutableArray *)points bot:(Bot *)bot{
48
49
50    if([points count] > 0){
51        NSValue *value = [points objectAtIndex:self.iterator];
52        CGPoint toGo = value.CGPointValue;
53
54        if (CGPointDistance(bot.position, toGo) < MAX_DISTANCE_TO_NODE) {
55            self.iterator++;
56            if (self.iterator == [points count]) {
57                self.iterator = points.count -1;
58            }
59        }
60
61        return [bot.steeringBehaviors seek:toGo andBot:bot];
62    }
63
64    return CGPointZero;
65}
66
67
68-(CGPoint) avoidanceCollision:(SKNode*) obstacle andBot:(Bot *)bot {
69
70    obstacle = (SKSpriteNode *) obstacle;
71
72    CGPoint obstacleCenter = CGPointMake([self centerX:obstacle], [self centerY:obstacle]);
73
74    CGPoint desiredVelocity = CGPointMultiplyScalar(CGPointNormalize(CGPointSubtract(bot.
              position, obstacleCenter)),MAX_SPEED);
75
76    return CGPointSubtract(desiredVelocity, bot.velocityPointsPerFrame);
77}
78
79
80
81@end
```

APÊNDICE C – Código da classe Graph

```
1 //
2 //  Graph.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 01/09/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "Graph.h"
10 #import "SKTUtils.h"
11
12 @interface Graph ()
13
14 @property (nonatomic,strong) NSMutableArray *positions;
15 @property (nonatomic,strong) NSMutableArray *edgesList;
16
17 @end
18
19 @implementation Graph
20
21 - (instancetype) initGraph {
22     self = [super init];
23     if (self) {
24
25         self.nodesList = [[NSMutableArray alloc] init];
26         self.edgesList = [[NSMutableArray alloc] init];
27         [self createGraph];
28         [self createConnections];
29     }
30     return self;
31 }
32
33 -(void) createGraph {
34
35     NSDictionary *nodesAndPositions;// = [[NSMutableDictionary alloc] init];
36     NSDictionary *nodesAndEdges;
37     NSString *positions;
38     NSString *edges;
39     NSArray *array;
40
41     NSString *filePath = [[NSBundle mainBundle] pathForResource:@"GraphPositions" ofType:@"
             plist"];
42     nodesAndPositions = [NSDictionary dictionaryWithContentsOfFile:filePath];
43
44     filePath = [[NSBundle mainBundle] pathForResource:@"GraphConnections" ofType:@"plist"];
45     nodesAndEdges = [NSDictionary dictionaryWithContentsOfFile:filePath];
46
47     for (int i = 1; i <= [nodesAndPositions count] ; i++){
48
49         NSString *key = [NSString stringWithFormat:@"%d",i];
50
```

```
51        positions = [nodesAndPositions objectForKey:key];
52
53        ListVertex *listVertex = [[ListVertex alloc] init];
54        listVertex.iD = i;
55
56        array = [positions componentsSeparatedByString:@","];
57        float x = [[array objectAtIndex:0] floatValue];
58        float y = [[array objectAtIndex:1] floatValue];
59
60        listVertex.position = CGPointMake(x, y);
61        [self.nodesList addObject:listVertex];
62
63        edges = [nodesAndEdges objectForKey:key];
64        array = [edges componentsSeparatedByString:@","];
65
66        [self.edgesList addObject:array];
67    }
68}
69
70-(void) createConnections {
71
72    ListVertex *currentList;
73    Node *currentNode, *previousNode;
74    NSArray *arrayTemp;
75
76    for (int i = 0; i < [self.edgesList count]; i++) {
77
78        arrayTemp = [self.edgesList objectAtIndex:i];
79        currentList = [self.nodesList objectAtIndex:i];
80
81        for(int j =0; j < [arrayTemp count]; j++){
82            float nodeFloat = [[arrayTemp objectAtIndex:j] floatValue];
83            currentNode = [[Node alloc] init];
84            currentNode.indicator = [self.nodesList objectAtIndex:(nodeFloat - 1)];
85            currentNode.edgeValue = CGPointDistance(currentList.position, currentNode.
                      indicator.position);
86
87            if (j == 0) {
88                currentList.next = currentNode;
89            }
90            else if (j > 0){
91                previousNode.next = currentNode;
92
93                if(j == ([arrayTemp count] - 1))
94                   currentNode.next = nil;
95
96            }
97            previousNode = currentNode;
98        }
99    }
100}
101
102@end
```

APÊNDICE D – Código da classe DetectEnemy

```
1 //
2 //  DetectEnemy.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 07/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "DetectEnemy.h"
10 #import "SKTUtils.h"
11
12 @implementation DetectEnemy
13
14 -(Player *) detectPlayersAndBot{
15
16     float distance = CGPointDistance(self.actualBot.position, self.actualBot.player.position
              );
17     if (distance < PANIC_MAX_DISTANCE && self.actualBot.player.team != self.actualBot.team
              && !self.actualBot.player.isDying) {
18         return self.actualBot.player;
19     }
20
21     for (Bot *botAround in self.actualBot.bots) {
22
23         float distance = CGPointDistance(self.actualBot.position, botAround.position);
24
25         if(botAround.team != self.actualBot.team) {
26
27             if (distance < PANIC_MAX_DISTANCE && !botAround.isDying) {
28                 return botAround;
29             }
30         }
31         else {
32             if (distance < PANIC_DISTANCE && !botAround.isDying) {
33                 [self.actualBot applySteeringForce:[self.actualBot.steeringBehaviors flee:
                          botAround.position andBot:self.actualBot]];
34             }
35         }
36     }
37     return nil;
38 }
39
40 @end
```

APÊNDICE E – Código da classe PathManager

```
1//
2//  PathManager.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 15/10/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "PathManager.h"
10#import "Graph.h"
11#import "ListVertex.h"
12#import "Node.h"
13#import "SKTUtils.h"
14
15@implementation PathManager
16
17-(NSMutableArray *) searchPathA:(Graph *)graph byRoot:(ListVertex *)rootNode andDestination
          :(ListVertex *) destination{
18
19    NSMutableArray *openList = [[NSMutableArray alloc] init];
20    ListVertex *currentNode;
21    ListVertex *successorNode;
22    Node *nodeTemp;
23    ListVertex *firstNode;
24    ListVertex *iterateNodeInGraph;
25    BOOL nodeAlreadyInOpenList = NO;
26
27    float x = INT_MAX;
28
29    if (destination == nil) {
30        if (((self.bot.team == kTeamRed) && (!self.bot.isHoldGem)) || (self.bot.team ==
                  kTeamBlue && self.bot.isHoldGem)) {
31            destination = [graph.nodesList objectAtIndex:0];
32        }
33        else {
34            destination = [graph.nodesList objectAtIndex:15];
35        }
36    }
37
38    if (destination.iD == rootNode.iD) {
39        return [self generateArray:destination isInBase:YES];
40    }
41
42    firstNode = [[ListVertex alloc] init];
43    firstNode.position = CGPointMake(rootNode.position.x, rootNode.position.x);
44    firstNode.iD = rootNode.iD;
45    firstNode.accumulatedCost = 0;
46    firstNode.parent = nil;
```

```
47  firstNode.finalCost = CGPointDistance(rootNode.position, destination.position);
48
49  [openList addObject:firstNode];
50
51  while ([openList count] > 0) {
52
53      for (ListVertex *nodeList in openList) {
54          if (nodeList.finalCost < x) {
55              currentNode = nodeList;
56              x = currentNode.finalCost;
57
58          }
59      }
60      x = INT_MAX;
61
62      [openList removeObject:currentNode];
63      if (currentNode.iD == destination.iD) {
64          return [self generateArray:currentNode isInBase:NO];
65      }
66
67      iterateNodeInGraph = [graph.nodesList objectAtIndex:currentNode.iD - 1];
68
69      for (nodeTemp = iterateNodeInGraph.next; nodeTemp != nil;) {
70
71          nodeAlreadyInOpenList = NO;
72
73          successorNode = [[ListVertex alloc] init];
74          successorNode.position = CGPointMake(nodeTemp.indicator.position.x, nodeTemp.
                    indicator.position.y);
75          successorNode.parent = currentNode;
76          successorNode.iD = nodeTemp.indicator.iD;
77          successorNode.accumulatedCost = currentNode.accumulatedCost + nodeTemp.edgeValue;
78          successorNode.finalCost = successorNode.accumulatedCost + CGPointDistance(
                    successorNode.position, destination.position);
79
80          for (ListVertex *listTemp in openList) {
81              if (successorNode.iD == listTemp.iD) {
82                  nodeAlreadyInOpenList = YES;
83                  if (listTemp.accumulatedCost > successorNode.accumulatedCost) {
84                      listTemp.parent = successorNode.parent;
85                      listTemp.accumulatedCost = successorNode.accumulatedCost;
86                      listTemp.finalCost = successorNode.finalCost;
87                  }
88              }
89          }
90          if (!nodeAlreadyInOpenList) {
91              [openList addObject:successorNode];
92          }
93
94          if (nodeTemp.next == nil) {
95              nodeTemp = nil;
96          } else {
97              nodeTemp = nodeTemp.next;
98          }
99      }
100 }
101 return nil;
```

```
102}
103
104-(NSMutableArray *) generateArray:(ListVertex *)node isInBase:(BOOL)isInBase {
105
106    NSMutableArray *listOfNodes = [[NSMutableArray alloc] init];
107
108    for (ListVertex *nodeTemp = node; ; ) {
109
110        CGPoint position = CGPointMake(nodeTemp.position.x, nodeTemp.position.y);
111        [listOfNodes addObject:[NSValue valueWithCGPoint:position]];
112
113        if(nodeTemp.parent == nil){
114            break;
115        }
116        else {
117            nodeTemp = nodeTemp.parent;
118        }
119    }
120    NSArray *newArray = [[listOfNodes reverseObjectEnumerator] allObjects];
121
122    NSMutableArray *mutableArray = [NSMutableArray arrayWithArray:newArray];
123    if (!isInBase) {
124        [mutableArray removeObjectAtIndex:0];
125    }
126    return mutableArray;
127}
128
129@end
```

APÊNDICE F – Código da classe DetectGem

```
1 //
2 //  DetectGem.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 20/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "DetectGem.h"
10 #import "SKTUtils.h"
11 #import "Gem.h"
12 #import "Graph.h"
13 #import "PathManager.h"
14
15 @implementation DetectGem
16
17 -(NSMutableArray *) detectBot:(Bot *) bot andGem:(Gem *)gem{
18     NSMutableArray *pathToGo;
19
20     float x = (float) INT16_MAX;
21     ListVertex *closerNode;
22
23     for (ListVertex *node in bot.graph.nodesList) {
24
25         float distance = CGPointDistance(node.position, gem.position);
26
27         if (distance < x){
28             x = distance;
29             closerNode = node;
30         }
31     }
32
33     int rootNodeID = [bot closerNode];
34     ListVertex *rootNode = [bot.graph.nodesList objectAtIndex:rootNodeID - 1];
35
36     pathToGo = [bot.pathManager searchPathA:bot.graph byRoot:rootNode andDestination:
                closerNode];
37
38     if (gem)
39         [pathToGo addObject:[NSValue valueWithCGPoint:gem.position]];
40
41     return pathToGo;
42
43 }
44
45 @end
```

APÊNDICE G – Código da classe BotAttacker

```
1//
2//  BotAttacker.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 25/08/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "BotAttacker.h"
10#import "SKTUtils/SKTUtils.h"
11#import "Bot.h"
12#import "SteeringBehaviors.h"
13#import "RetangleCollision.h"
14#import "TakeGemToBase.h"
15
16#define MASS 1
17
18@implementation BotAttacker
19
20-(void) initActions {
21
22    self.currentState = [[SearchEnemyGem alloc] init];
23    [self.currentState enter:self];
24
25}
26
27-(void) changeEstate:(id <StateBot>)newState{
28
29    [self.currentState exit:self];
30
31    self.previousState = self.currentState;
32
33    self.currentState = newState;
34
35    [self.currentState enter:self];
36}
37
38-(void) revertToPreviousState {
39
40    [self changeEstate:self.previousState];
41
42}
43
44-(void) update {
45
46    if (self.currentState && self.isInView) {
47
48        [self verifyIfBotHasTheGem];
```

```
49
50        [self.currentState execute:self];
51
52    }
53
54    [self calculateVelocityPointsPerFrame];
55    [self.steeringBehaviors calculateRetanglePosition:self.retangle andBot:self];
56
57}
58
59-(void) verifyIfBotHasTheGem {
60    if (self.isHoldGem) {
61
62        if (self.teamGem == self.team) {
63            self.allyGemWasStolen = NO;
64        }
65
66        id <StateBot> nextState = [[TakeGemToBase alloc] init];
67        [self changeEstate:nextState];
68    }
69}
70
71@end
```

APÊNDICE H – Código da classe BotDefender

```
1 //
2 //  BotDefender.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 29/09/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "BotDefender.h"
10 #import "PatrollingBase.h"
11 #import "ReturnToBase.h"
12
13 @implementation BotDefender
14
15 -(void) initActions {
16
17     self.currentState = [[PatrollingBase alloc] init];
18     [self.currentState enter:self];
19
20 }
21
22 -(void) changeEstate:(id <StateBot>)newState{
23
24     [self.currentState exit:self];
25
26     self.previousState = self.currentState;
27
28     self.currentState = newState;
29
30     [self.currentState enter:self];
31 }
32
33 -(void) revertToPreviousState {
34
35     [self changeEstate:self.previousState];
36
37 }
38
39 -(void) update {
40
41     if (self.currentState && self.isInView) {
42
43         [self verifyIfBotHasTheGem];
44
45         [self.currentState execute:self];
46     }
47
48     [self calculateVelocityPointsPerFrame];
```

```
49    [self.steeringBehaviors calculateRetanglePosition:(SKSpriteNode *)self.retangle andBot:
              self];
50
51}
52
53-(void) verifyIfBotHasTheGem {
54    if (self.isHoldGem) {
55
56        if (self.teamGem == self.team) {
57            self.allyGemWasStolen = NO;
58        }
59
60        id <StateBot> nextState = [[ReturnToBase alloc] init];
61        [self changeEstate:nextState];
62    }
63}
64
65@end
```

APÊNDICE I – Código da classe SearchEnemyGem

```
1//
2//  SeachEnemyGem.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 11/09/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "SearchEnemyGem.h"
10#import "SKTUtils.h"
11#import "Attack.h"
12#import "DetectEnemy.h"
13#import "TakeGemToBase.h"
14#import "Gem.h"
15#import "DetectGem.h"
16#import "ProtectTheGemCarrier.h"
17
18@interface SearchEnemyGem ()
19
20@property (nonatomic,strong) NSMutableArray *currentPath;
21@property (nonatomic) int iterator;
22@property id <StateBot> currentState;
23
24@end
25
26@implementation SearchEnemyGem
27
28-(void) enter:(Bot *)bot {
29
30    NSDictionary *nodesToGo;
31    NSString *positions;
32    NSArray *array;
33    NSMutableArray *predefinedPaths = [[NSMutableArray alloc] init];
34    self.currentPath = [[NSMutableArray alloc] init];
35    self.iterator = 0;
36    bot.steeringBehaviors.iterator = 0;
37
38    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"AttackerRoutes" ofType:@"
            plist"];
39    nodesToGo = [NSDictionary dictionaryWithContentsOfFile:filePath];
40
41    if (bot.team == kTeamBlue) {
42
43        for (int i = 1; i<= [nodesToGo count]; i++) {
44
45            NSString *key = [NSString stringWithFormat:@"%d",i];
46            positions = [nodesToGo objectForKey:key];
47
```

```
48          array = [positions componentsSeparatedByString:@","];
49          [predefinedPaths addObject:array];
50      }
51  }
52  else if (bot.team == kTeamRed) {
53      for (int i = [nodesToGo count]; i>= 1; i--) {
54
55          NSString *key = [NSString stringWithFormat:@"%d",i];
56          positions = [nodesToGo objectForKey:key];
57
58          array = [positions componentsSeparatedByString:@","];
59          NSArray *newArray = [[array reverseObjectEnumerator] allObjects];
60          [predefinedPaths addObject:newArray];
61      }
62  }
63  float randomPath = arc4random()%[predefinedPaths count];
64  [self followPath:[predefinedPaths objectAtIndex:randomPath] andBot:bot];
65  [self applyPathDifficulty:bot];
66}
67
68-(void) execute:(Bot *)bot {
69  //If the enemy gem is out the enemy base, in the ground
70  if(bot.enemyGem && !bot.enemyGem.isInBase ) {
71      self.currentPath = [bot.detectGem detectBot:bot andGem:bot.enemyGem];
72
73  }
74
75  [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot]];
76
77  bot.enemy = [bot.detectEnemy detectPlayersAndBot];
78
79  if (bot.enemy) {
80      id <StateBot> nextState = [[Attack alloc] init];
81      [bot changeEstate:nextState];
82  }
83
84  else if (bot.enemyGemWasStolen && !bot.isHoldGem && bot.allyThief) {
85      id <StateBot> nextState = [[ProtectTheGemCarrier alloc] init];
86      [bot changeEstate:nextState];
87  }
88}
89
90-(void) exit:(Bot *)bot {
91  bot.steeringBehaviors.iterator = 0;
92}
93
94-(void) followPath:(NSArray *)positions andBot:(Bot *)bot{
95
96  for (NSString *nodeStringIndicator in positions) {
97      float x = [nodeStringIndicator floatValue];
98      ListVertex *node = [bot.graph.nodesList objectAtIndex:(x - 1)];
99      CGPoint position = CGPointMake(node.position.x, node.position.y);
100     [self.currentPath addObject:[NSValue valueWithCGPoint:position]];
101 }
102}
103
104-(void) applyPathDifficulty:(Bot *)bot {
```

```
105
106     switch (bot.difficulty) {
107         //If the game is in the hard level and the bot has the same team of player, so the
                    player allies take a longer time to reach the enemy's base.
108         case kHard:
109             if (bot.team == bot.player.team) {
110                 [self changeCurrentPath];
111             }
112             break;
113         default:
114             //If the game is in the easy level and the bot has the same team of player, so
                        the player allies do not take much time to reach the enemy's base.
115             if (bot.team != bot.player.team) {
116                 [self changeCurrentPath];
117             }
118             break;
119     }
120 }
121
122 -(void) changeCurrentPath {
123      NSMutableArray *tempArray = [[NSMutableArray alloc] init];
124
125     for (int j = 0; j < [self.currentPath count]; j++) {
126         NSValue *point = [self.currentPath objectAtIndex:j];
127         [tempArray addObject:point];
128         if(j == 3) {
129             [tempArray addObject:[self.currentPath objectAtIndex:2]];
130             [tempArray addObject:[self.currentPath objectAtIndex:1]];
131             [tempArray addObject:[self.currentPath objectAtIndex:2]];
132             [tempArray addObject:[self.currentPath objectAtIndex:3]];
133         }
134     }
135     self.currentPath = tempArray;
136 }
137
138 @end
```

APÊNDICE J – Código da classe Attack

```
1 //
2 //  Attack.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 30/09/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "Attack.h"
10 #import "SteeringBehaviors.h"
11 #import "SKTUtils.h"
12 #import "SearchEnemyGem.h"
13 #import "PathManager.h"
14 #import "BestPathToBase.h"
15 #import "BotDefender.h"
16 #import "Gem.h"
17 #import "SearchAllyGem.h"
18 #import "ProtectTheGemCarrier.h"
19 #import "TakeGemToBase.h"
20 #import "ReturnToBase.h"
21
22 @implementation Attack
23
24 -(void) enter:(Bot *)bot {
25     bot.isAttacking = YES;
26 }
27
28 -(void) execute:(Bot *)bot {
29
30     float distance = CGPointDistance(bot.position, bot.enemy.position);
31
32     if (distance > PANIC_MAX_DISTANCE || bot.enemy.isDying ){
33
34         if (bot.allyThief) {
35             [self applyDifficulty:bot];
36         }
37
38         id <StateBot> nextState = [[BestPathToBase alloc] init];
39         [bot changeEstate:nextState];
40     }
41
42     else if (distance < PANIC_DISTANCE ) {
43         [bot applySteeringForce:[bot.steeringBehaviors flee:bot.enemy.position andBot:bot]];
44     }
45
46     else if (distance > PANIC_DISTANCE && distance < PANIC_MAX_DISTANCE) {
47         [bot applySteeringForce:[bot.steeringBehaviors seek:bot.enemy.position andBot:bot]];
48     }
49 }
50
51 -(void)applyDifficulty:(Bot *)bot {
```

```
52
53    if (bot.difficulty != kHard) {
54        id <StateBot> nextState = [[ReturnToBase alloc] init];
55        [bot changeEstate:nextState];
56    }
57    else {
58        id <StateBot> nextState = [[ProtectTheGemCarrier alloc] init];
59        [bot changeEstate:nextState];
60    }
61}
62
63-(void) exit:(Bot *)bot{
64    bot.isAttacking = NO;
65    [bot stopShooting];
66}
67
68@end
```

APÊNDICE K – Código da classe BestPathToBase

```
1//
2//  BestPathToBase.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 19/10/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "BestPathToBase.h"
10#import "SKTUtils.h"
11#import "PathManager.h"
12#import "Bot.h"
13#import "DetectEnemy.h"
14#import "Attack.h"
15#import "BotAttacker.h"
16#import "Gem.h"
17#import "SearchAllyGem.h"
18#import "BotDefender.h"
19#import "AttackDefender.h"
20#import "ProtectTheGemCarrier.h"
21#import "TakeGemToBase.h"
22
23@implementation BestPathToBase
24
25-(void) enter:(Bot *)bot {
26
27    self.pathToGo = [[NSMutableArray alloc] init];
28
29    int rootNodeID = [bot closerNode];
30    ListVertex *node = [bot.graph.nodesList objectAtIndex:rootNodeID - 1];
31    self.pathToGo = [bot.pathManager searchPathA:bot.graph byRoot:node andDestination:nil];
32
33    bot.steeringBehaviors.iterator = 0;
34}
35
36-(void) execute:(Bot *)bot {
37
38    [bot applySteeringForce:[bot.steeringBehaviors followPath:self.pathToGo bot:bot]];
39    bot.enemy = [bot.detectEnemy detectPlayersAndBot];
40
41    if( [bot isKindOfClass:[BotAttacker class]]) {
42        if(bot.isHoldGem){
43            id <StateBot> nextState = [[TakeGemToBase alloc] init];
44            [bot changeEstate:nextState];
45        }
46
47        if(bot.enemy && !bot.isHoldGem){
48            id <StateBot> nextState = [[Attack alloc] init];
```

```
49            [bot changeEstate:nextState];
50        }
51
52        if ((bot.enemyGemWasStolen && !bot.isHoldGem && bot.difficulty != kHard)) {
53            id <StateBot> nextState = [[ProtectTheGemCarrier alloc] init];
54            [bot changeEstate:nextState];
55        }
56    }
57
58    else if ([bot isKindOfClass:[BotDefender class]]) {
59        if (bot.allyGemWasStolen) {
60            id <StateBot> nextState = [[SearchAllyGem alloc] init];
61            [bot changeEstate:nextState];
62        }
63    }
64
65    if(bot.isHoldGem) {
66        id <StateBot> nextState = [[TakeGemToBase alloc] init];
67        [bot changeEstate:nextState];
68    }
69}
70
71-(void) exit:(Bot *)bot{
72
73}
74
75@end
```

APÊNDICE L – Código da classe ProtectTheGem

```
1 //
2 //  ProtectTheGemCarrier.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 22/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "ProtectTheGemCarrier.h"
10 #import "Attack.h"
11 #import "DetectGem.h"
12 #import "SearchEnemyGem.h"
13 #import "DetectEnemy.h"
14 #import "TakeGemToBase.h"
15
16 @interface ProtectTheGemCarrier ()
17 @property (nonatomic,strong) NSMutableArray *currentPath;
18 @end
19
20 @implementation ProtectTheGemCarrier
21
22 -(void) enter:(Bot *)bot {
23     if (bot.difficulty == kHard) {
24         id <StateBot> nextState = [[TakeGemToBase alloc] init];
25         [bot changeEstate:nextState];
26     }
27 }
28
29 -(void) execute:(Bot *)bot {
30     [bot applySteeringForce:[bot.steeringBehaviors seek:bot.allyThief.position andBot:bot]];
31     bot.enemy = [bot.detectEnemy detectPlayersAndBot];
32
33     if (bot.enemy) {
34         id <StateBot> nextState = [[Attack alloc] init];
35         [bot changeEstate:nextState];
36     }
37     //If the enemy's gem is out of base, in the ground
38     if(bot.enemyGem && !bot.enemyGem.isInBase ) {
39         id <StateBot> nextState = [[SearchEnemyGem alloc] init];
40         [bot changeEstate:nextState];
41     }
42 }
43
44 -(void) exit:(Bot *)bot {
45 }
46
47 @end
```

APÊNDICE M – Código da classe TakeGemToBase

```
1 //
2 //  TakeEnemyGemToBase.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 20/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "TakeGemToBase.h"
10 #import "DetectEnemy.h"
11 #import "Attack.h"
12 #import "SearchEnemyGem.h"
13 #import "SKTUtils.h"
14 #import "PathManager.h"
15 #import "Bot.h"
16
17 @interface TakeGemToBase ()
18
19 @property (nonatomic,strong) NSMutableArray *currentPath;
20 @property (nonatomic) int iterator;
21 @property (nonatomic, retain) ListVertex *baseNode;
22
23 @end
24
25 @implementation TakeGemToBase
26
27 -(void) enter:(Bot *)bot {
28     NSDictionary *nodesToGo;
29     NSString *positions;
30     NSArray *array;
31     NSMutableArray *predefinedPaths = [[NSMutableArray alloc] init];
32     self.currentPath = [[NSMutableArray alloc] init];
33     self.iterator = 0;
34     float randomPath;
35
36     NSString *filePath = [[NSBundle mainBundle] pathForResource:@"AttackerRoutes" ofType:@"
            plist"];
37     nodesToGo = [NSDictionary dictionaryWithContentsOfFile:filePath];
38
39     if (bot.team == kTeamRed) {
40         self.baseNode = [bot.graph.nodesList objectAtIndex:15];
41
42         for (int i = 1; i<= [nodesToGo count]; i++) {
43             NSString *key = [NSString stringWithFormat:@"%d",i];
44             positions = [nodesToGo objectForKey:key];
45             array = [positions componentsSeparatedByString:@","];
46             [predefinedPaths addObject:array];
47         }
```

```
48    }
49    else if (bot.team == kTeamBlue) {
50        self.baseNode = [bot.graph.nodesList objectAtIndex:0];
51
52        for (int i = [nodesToGo count]; i>= 1; i--) {
53
54            NSString *key = [NSString stringWithFormat:@"%d",i];
55            positions = [nodesToGo objectForKey:key];
56            array = [positions componentsSeparatedByString:@","];
57            NSArray *newArray = [[array reverseObjectEnumerator] allObjects];
58            [predefinedPaths addObject:newArray];
59        }
60    }
61
62    if(CGPointDistance(bot.position, self.baseNode.position) > 100) {
63        PathManager *manager = [[PathManager alloc] init];
64        int rootNodeID = [bot closerNode];
65        ListVertex *node = [bot.graph.nodesList objectAtIndex:rootNodeID - 1];
66        NSMutableArray *path = [manager searchPathA:bot.graph byRoot:node andDestination:
                  self.baseNode];
67        self.currentPath = path;
68    }
69    else {
70        randomPath = arc4random()%[predefinedPaths count];
71        [self followPath:[predefinedPaths objectAtIndex:randomPath] andBot:bot];
72        [self applyDifficulty:bot];
73    }
74}
75
76-(void) execute:(Bot *)bot {
77
78    float distance = CGPointDistance(bot.position, self.baseNode.position);
79    [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot]];
80    bot.enemy = [bot.detectEnemy detectPlayersAndBot];
81
82    if (!bot.isHoldGem && !bot.allyThief) {
83        id <StateBot> nextState = [[SearchEnemyGem alloc] init];
84        [bot changeEstate:nextState];
85    }
86
87    if (distance < 100 && !bot.isHoldGem) {
88        [bot revertToPreviousState];
89    }
90    else if (distance < 25 && bot.isHoldGem) {
91        [bot returnGem];
92        [bot revertToPreviousState];
93    }
94}
95
96-(void) exit:(Bot *)bot {
97    bot.steeringBehaviors.iterator = 0;
98}
99
100-(void) followPath:(NSArray *)positions andBot:(Bot *)bot{
101
102    for (NSString *nodeStringIndicator in positions) {
103        float x = [nodeStringIndicator floatValue];
```

```
104         ListVertex *node = [bot.graph.nodesList objectAtIndex:(x - 1)];
105         CGPoint position = CGPointMake(node.position.x, node.position.y);
106         [self.currentPath addObject:[NSValue valueWithCGPoint:position]];
107     }
108 }
109 
110 -(void) applyDifficulty:(Bot *)bot {
111 
112     NSMutableArray *tempArray = [[NSMutableArray alloc] init];
113     switch (bot.difficulty) {
114 
115         case kEasy:
116             for (int j = 0; j < [self.currentPath count]; j++) {
117                 NSValue *point = [self.currentPath objectAtIndex:j];
118                 [tempArray addObject:point];
119                 if(j == 3) {
120                     [tempArray addObject:[self.currentPath objectAtIndex:2]];
121                     [tempArray addObject:[self.currentPath objectAtIndex:3]];
122                 }
123             }
124             self.currentPath = tempArray;
125             break;
126         default:
127             break;
128     }
129 }
130 
131 @end
```

APÊNDICE N – Código da classe PatrollingBase

```
1//
2//  PatrollingBase.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 29/09/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "PatrollingBase.h"
10#import "SteeringBehaviors.h"
11#import "PatrollingBase.h"
12#import "Graph.h"
13#import "SKTUtils.h"
14#import "AttackDefender.h"
15#import "Bot.h"
16#import "DetectEnemy.h"
17#import "Gem.h"
18#import "SearchAllyGem.h"
19
20#define RANGE_OF_WANDER 50
21
22@implementation PatrollingBase
23
24-(void) enter:(Bot *)bot {
25
26    NSDictionary *positionsToGo;
27    NSString *positionsString;
28    NSArray *array;
29
30    self.currentPath = [[NSMutableArray alloc] init];
31
32    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"DefenderRoutes" ofType:@"
              plist"];
33    positionsToGo = [NSDictionary dictionaryWithContentsOfFile:filePath];
34
35    if (bot.team == kTeamBlue) {
36        for (int i = 0; i < 3; i++){
37            NSString *key = [NSString stringWithFormat:@"%d",i];
38            positionsString = [positionsToGo objectForKey:key];
39            array = [positionsString componentsSeparatedByString:@","];
40            float x = [[array objectAtIndex:0] floatValue];
41            float y = [[array objectAtIndex:1] floatValue];
42            CGPoint position = CGPointMake(x, y);
43            [self.currentPath addObject:[NSValue valueWithCGPoint:position]];
44        }
45    }
46    else if (bot.team == kTeamRed){
47        for (int i = 3; i < 6; i++){
```

```
48            NSString *key = [NSString stringWithFormat:@"%d",i];
49            positionsString = [positionsToGo objectForKey:key];
50            array = [positionsString componentsSeparatedByString:@","];
51            float x = [[array objectAtIndex:0] floatValue];
52            float y = [[array objectAtIndex:1] floatValue];
53            CGPoint position = CGPointMake(x, y);
54            [self.currentPath addObject:[NSValue valueWithCGPoint:position]];
55        }
56    }
57    [self shuffle];
58}
59
60-(void) execute:(Bot *)bot {
61    [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot]];
62
63    if (bot.steeringBehaviors.iterator == [self.currentPath count] - 1) {
64        [self shuffle];
65        bot.steeringBehaviors.iterator = 0;
66    }
67    bot.enemy = [bot.detectEnemy detectPlayersAndBot];
68    if (bot.enemy) {
69        id <StateBot> nextState = [[AttackDefender alloc] init];
70        [bot changeEstate:nextState];
71    }
72    if (bot.allyGemWasStolen) {
73        id <StateBot> nextState = [[SearchAllyGem alloc] init];
74        [bot changeEstate:nextState];
75    }
76}
77
78-(void) exit:(Bot *)bot {
79}
80
81-(void) shuffle {
82    NSUInteger count = [self.currentPath count];
83    for (int i = 0; i < count; ++i) {
84        NSUInteger nElements = count - i;
85        int n = (arc4random() % nElements) + i;
86        [self.currentPath exchangeObjectAtIndex:i withObjectAtIndex:n];
87    }
88}
89
90@end
```

APÊNDICE O – Código da classe AttackDefender

```
1 //
2 //  AttackDefender.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 21/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "AttackDefender.h"
10 #import "Gem.h"
11 #import "BestPathToBase.h"
12 #import "ReturnToBase.h"
13 #import "SKTUtils.h"
14 #import "SearchAllyGem.h"
15
16 @implementation AttackDefender
17
18 -(void) enter:(Bot *)bot {
19     bot.isAttacking = YES;
20 }
21
22 -(void) execute:(Bot *)bot {
23     float distance = CGPointDistance(bot.position, bot.enemy.position);
24     if (distance > PANIC_MAX_DISTANCE || bot.enemy.isDying){
25         [bot revertToPreviousState];
26     }
27     else if (distance < PANIC_DISTANCE ) {
28         [bot applySteeringForce:[bot.steeringBehaviors flee:bot.enemy.position andBot:bot]];
29     }
30     else if (distance > PANIC_DISTANCE && distance < PANIC_MAX_DISTANCE) {
31         [bot applySteeringForce:[bot.steeringBehaviors seek:bot.enemy.position andBot:bot]];
32     }
33 }
34
35 -(void) exit:(Bot *)bot {
36     bot.isAttacking = NO;
37     [bot stopShooting];
38 }
39
40 @end
```

APÊNDICE P – Código da classe SearchAllyGem

```
1 //
2 //  SearchAllyGem.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 21/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8 #import "SearchAllyGem.h"
9 #import "PathManager.h"
10 #import "DetectGem.h"
11 #import "BestPathToBase.h"
12 #import "DetectEnemy.h"
13 #import "AttackDefender.h"
14 #import "ReturnToBase.h"
15
16 @interface SearchAllyGem ()
17 @property (nonatomic,strong) NSMutableArray *currentPath;
18 @end
19
20 @implementation SearchAllyGem
21
22 -(void) enter:(Bot *)bot {
23     bot.steeringBehaviors.iterator = 0;
24
25 }
26
27 -(void) execute:(Bot *)bot {
28
29     bot.enemy = [bot.detectEnemy detectPlayersAndBot];
30
31     if (bot.allyGemWasStolen && !bot.enemyThief){
32         self.currentPath = [bot.detectGem detectBot:bot andGem:bot.allyGem];
33         [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot
                 ]];
34     }
35     else if (bot.enemy) {
36         //The defensor bot attacks only the one that carries the gem and do not distract
                 with other enemies
37         if (bot.difficulty == kHard) {
38             if (bot.enemy == bot.enemyThief) {
39                 id <StateBot> nextState = [[AttackDefender alloc] init];
40                 [bot changeEstate:nextState];
41             }
42         }
43         else{
44             id <StateBot> nextState = [[AttackDefender alloc] init];
45             [bot changeEstate:nextState];
46         }
```

```
47    }
48    else if(bot.allyGemWasStolen && bot.enemyThief) {
49        [bot applySteeringForce:[bot.steeringBehaviors seek:bot.enemyThief.position andBot:
                bot]];
50    }
51
52    if (bot.isHoldGem || bot.allyGemWasStolen == NO) {
53        bot.allyGemWasStolen = NO;
54        id <StateBot> nextState = [[ReturnToBase alloc] init];
55        [bot changeEstate:nextState];
56    }
57}
58
59-(void) exit:(Bot *)bot {
60    bot.steeringBehaviors.iterator = 0;
61}
62
63@end
```

APÊNDICE Q – Código da classe ReturnToBase

```
1//
2//  ReturnToBase.m
3//  LostGemsFinal
4//
5//  Created by Bruno Rodrigues de Andrade on 21/10/15.
6//  Copyright (c) 2015 Henrique Santos. All rights reserved.
7//
8
9#import "ReturnToBase.h"
10#import "PathManager.h"
11#import "DetectEnemy.h"
12#import "AttackDefender.h"
13#import "SKTUtils.h"
14#import "PatrollingBase.h"
15#import "Gem.h"
16#import "SearchAllyGem.h"
17#import "BotAttacker.h"
18#import "Attack.h"
19
20@interface ReturnToBase ()
21@property (nonatomic, retain) NSMutableArray *currentPath;
22@property (nonatomic, retain) ListVertex *baseNode;
23@end
24
25@implementation ReturnToBase
26
27-(void) enter:(Bot *)bot {
28    if (bot.team == kTeamRed) {
29        self.baseNode = [bot.graph.nodesList objectAtIndex:15];
30    }
31    else {
32        self.baseNode = [bot.graph.nodesList objectAtIndex:0];
33    }
34    PathManager *manager = [[PathManager alloc] init];
35    int rootNodeID = [bot closerNode];
36    ListVertex *node = [bot.graph.nodesList objectAtIndex:rootNodeID - 1];
37    NSMutableArray *path = [manager searchPathA:bot.graph byRoot:node andDestination:self.
              baseNode];
38    self.currentPath = path;
39    bot.steeringBehaviors.iterator = 0;
40}
41
42-(void) execute:(Bot *)bot {
43    if (bot.allyGemWasStolen) {
44        // Goes back to SearchAllyGem
45        [bot revertToPreviousState];
46    }
47    float distance = CGPointDistance(bot.position, self.baseNode.position);
```

```
48
49    if ((distance < 100 && !bot.isHoldGem) || (distance < 50 && bot.isHoldGem)) {
50        if (distance < 25 && bot.isHoldGem) {
51            [bot returnGem];
52        }
53        id <StateBot> nextState = [[PatrollingBase alloc] init];
54        [bot changeEstate:nextState];
55    }
56    [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot]];
57    bot.enemy = [bot.detectEnemy detectPlayersAndBot];
58    if (!bot.isHoldGem && bot.enemy) {
59        if ([bot isKindOfClass:[BotAttacker class]]) {
60            id <StateBot> nextState = [[Attack alloc] init];
61            [bot changeEstate:nextState];
62        }
63        else{
64            id <StateBot> nextState = [[AttackDefender alloc] init];
65            [bot changeEstate:nextState];
66        }
67    }
68}
69
70-(void) exit:(Bot *)bot {
71    bot.steeringBehaviors.iterator = 0;
72}
73
74@end
```

APÊNDICE R – Código da classe FollowPlayer

```
1 //
2 //  FollowPlayer.m
3 //  LostGemsFinal
4 //
5 //  Created by Bruno Rodrigues de Andrade on 31/10/15.
6 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
7 //
8
9 #import "FollowPlayer.h"
10 #import "PathManager.h"
11 #import "SKTUtils.h"
12 #import "SteeringBehaviors.h"
13 #import "DetectEnemy.h"
14 #import "AttackDefender.h"
15 #import "TakeGemToBase.h"
16 #import "ReturnToBase.h"
17
18 @interface FollowPlayer ()
19
20 @property (nonatomic,strong) NSMutableArray *currentPath;
21
22 @end
23
24 @implementation FollowPlayer
25
26 -(void) enter:(Bot *)bot {
27
28     bot.steeringBehaviors.iterator = 0;
29
30 }
31
32
33 -(void) execute:(Bot *)bot {
34
35     if (bot.player.isDying) {
36         id <StateBot> nextState = [[ReturnToBase alloc] init];
37         [bot changeEstate:nextState];
38     }
39
40     bot.enemy = [bot.detectEnemy detectPlayersAndBot];
41
42     float distance = CGPointDistance(bot.position, bot.player.position);
43     if (distance > PANIC_MAX_DISTANCE + 50) {
44         bot.steeringBehaviors.iterator = 0;
45         self.currentPath = [self detectPlayer:bot];
46         [bot applySteeringForce:[bot.steeringBehaviors followPath:self.currentPath bot:bot
                 ]];
47     }
```

```
48    else if (distance < PANIC_DISTANCE/3 && !bot.player.isDying) {
49        [bot applySteeringForce:[bot.steeringBehaviors flee:bot.player.position andBot:bot
                  ]];
50    }
51
52    else {
53        [bot applySteeringForce:[bot.steeringBehaviors seek:bot.player.position andBot:bot
                  ]];
54    }
55
56
57
58
59    if (bot.enemy) {
60        id <StateBot> nextState = [[AttackDefender alloc] init];
61        [bot changeEstate:nextState];
62    }
63
64}
65
66-(void) exit:(Bot *)bot {
67    bot.steeringBehaviors.iterator = 0;
68}
69
70
71
72-(NSMutableArray *) detectPlayer:(Bot *)bot {
73
74    NSMutableArray *pathToGo;
75    float x = (float) INT16_MAX;
76    ListVertex *closerNode;
77
78    for (ListVertex *node in bot.graph.nodesList) {
79
80        float distance = CGPointDistance(node.position, bot.player.position);
81
82        if (distance < x){
83            x = distance;
84            closerNode = node;
85        }
86    }
87
88    int rootNodeID = [bot closerNode];
89    ListVertex *rootNode = [bot.graph.nodesList objectAtIndex:rootNodeID - 1];
90
91    pathToGo = [bot.pathManager searchPathA:bot.graph byRoot:rootNode andDestination:
                closerNode];
92
93    return pathToGo;
94
95}
96
97@end
```

APÊNDICE S – Código da classe TestAttack

```
1 //  TestSearchEnemyGem.m
2 //  LostGemsFinal
3 //
4 //  Created by Bruno Rodrigues de Andrade on 27/10/15.
5 //  Copyright (c) 2015 Henrique Santos. All rights reserved.
6
7 #import <UIKit/UIKit.h>
8 #import <XCTest/XCTest.h>
9 #import "BotAttacker.h"
10 #import "GameLayer.h"
11 #import "SKTUtils.h"
12 #import "ListVertex.h"
13 #import "BestPathToBase.h"
14 #import "Gem.h"
15 #import "GameLayerMultiplayer.h"
16 #import "Player.h"
17 #import "SharedTextureCache.h"
18 #import "Attack.h"
19 #import "ReturnToBase.h"
20 #import "TakeGemToBase.h"
21 #import "ProtectTheGemCarrier.h"
22
23 #define WIDTH 568
24 #define HEIGTH 320
25
26 @interface TestAttack : XCTestCase
27
28 @property (nonatomic, strong) BotAttacker *bot;
29 @property (nonatomic, strong) GameLayerMultiplayer *gameLayer;
30 @property (nonatomic,strong) Gem *gem;
31 @property id <StateBot> firstState;
32
33 @end
34
35 @implementation TestAttack
36
37 - (void)setUp {
38     [super setUp];
39     self.gameLayer =[[GameLayerMultiplayer alloc] initWithSize:CGSizeMake(WIDTH, HEIGTH)
              andTypePlayer:ELF andTeam:kTeamBlue];
40     [self.gameLayer createBlueBots:5 andRedBots:5];
41     self.bot = [self.gameLayer.bots objectAtIndex:1];
42     self.firstState = [[Attack alloc] init];
43     [self.bot changeEstate:self.firstState];
44     self.bot.enemy = self.bot.player;
45
46     self.gem = [[Gem alloc]initWithTexture:[[SharedTextureCache sharedCache]textureNamed:@"
              GemBlue.png"] andTeam:kTeamBlue];
```

```
47}
48
49-(void) testExecuteMethod {
50    [self.bot.currentState execute:self.bot];
51    XCTAssertEqualObjects(self.firstState, self.bot.currentState);
52}
53
54-(void)testIfTheBotSeekTheEnemyIfBotIsInTheRangeOfPanicDistance {
55    self.bot.player.position = CGPointMake(self.bot.position.x + PANIC_DISTANCE + 1, self.
              bot.position.y);
56    [self.bot.currentState execute:self.bot];
57    XCTAssertEqualObjects(self.firstState, self.bot.currentState);
58}
59
60-(void) testIfTheBotFleeIfBotIsTooClose {
61    self.bot.player.position = CGPointMake(self.bot.position.x + 1 , self.bot.position.y);
62    [self.bot.currentState execute:self.bot];
63    XCTAssertEqualObjects(self.firstState, self.bot.currentState);
64}
65
66-(void)testIfTheBotTakesTheBetterWayToEnemyBaseWhenTheEnemyIsFar {
67    self.bot.player.position = CGPointMake(self.bot.position.x + PANIC_MAX_DISTANCE + 1,
              self.bot.position.y);
68    [self.bot.currentState execute:self.bot];
69    XCTAssertTrue([self.bot.currentState isKindOfClass:[BestPathToBase class]]);
70}
71
72-(void) testIfBotGoesBackToBaseWhenKillsEnemyAndTheGemWasStolen {
73    self.bot.allyThief = [self.gameLayer.bots objectAtIndex:1];
74    self.bot.player.position = CGPointMake(self.bot.position.x + PANIC_MAX_DISTANCE + 1,
              self.bot.position.y);
75    Bot *enemyBot = [self.gameLayer.bots objectAtIndex:5];
76    self.bot.position = CGPointMake(enemyBot.position.x, enemyBot.position.y);
77    self.bot.difficulty = kEasy;
78    [self.bot.currentState execute:self.bot];
79    XCTAssertTrue([self.bot.currentState isKindOfClass:[TakeGemToBase class]]);
80}
81
82-(void)testIfTheBotProtectTheGemCarrier {
83    self.bot.player.position = CGPointMake(self.bot.position.x + PANIC_MAX_DISTANCE + 1,
              self.bot.position.y);
84    self.bot.enemyGemWasStolen = YES;
85    self.bot.isHoldGem = NO;
86    self.bot.difficulty = kEasy;
87    [self.bot.currentState execute:self.bot];
88    XCTAssertTrue([self.bot.currentState isKindOfClass:[BestPathToBase class]]);
89}
90
91@end
```

APÊNDICE T – Código Teste SteeringBehaviours

```
1//
2//  SteeringBehaviors.m
3//  Network test Multipeer
4//
5//  Created by Bruno Rodrigues de Andrade on 12/06/15.
6//  Copyright (c) 2015 Bruno Rodrigues de Andrade. All rights reserved.
7//
8
9#import "SteeringBehaviors.h"
10#import "RedSquare.h"
11
12@implementation SteeringBehaviors
13
14- (instancetype)init
15{
16    self = [super init];
17    if (self) {
18        self.wanderTarget = CGPointMake(0, 0);
19    }
20    return self;
21}
22
23-(CGPoint) seek:(CGPoint)targetPos {
24
25    CGPoint desiredVelocity = CGPointMultiplyScalar( CGPointNormalize(CGPointSubtract(
              targetPos, self.redSquare.position)),self.redSquare.maxSpeed);
26
27    return CGPointSubtract(desiredVelocity, self.redSquare.vectorVelocity);
28}
29
30-(CGPoint) flee:(CGPoint)targetPos {
31    double panicDistance = 100;
32
33    if(CGPointDistancePoints(self.redSquare.position, targetPos) > panicDistance){
34        return CGPointMake(0, 0);
35    }
36
37    CGPoint desiredVelocity = CGPointMultiplyScalar(CGPointNormalize(CGPointSubtract(self.
              redSquare.position, targetPos)),self.redSquare.maxSpeed) ;
38
39    return CGPointSubtract(desiredVelocity, self.redSquare.vectorVelocity);
40}
41
42-(CGPoint) arrive:(CGPoint)targetPos andDeceleration:(float)slowingRadius {
43
44    CGPoint desiredVelocity;
45
46    CGPoint toTarget = CGPointSubtract(targetPos, self.redSquare.position);
```

```
47
48    double dist = CGPointLength(toTarget);
49
50    if(dist < slowingRadius ) {
51        double speed = dist/slowingRadius;
52        speed = [self truncateSpeed:speed andVelocity:self.redSquare.maxSpeed];
53        desiredVelocity = CGPointMultiplyScalar(CGPointMultiplyScalar(CGPointNormalize(
                  toTarget), self.redSquare.maxSpeed), speed);
54    }
55
56    else {
57        [self seek:targetPos];
58        desiredVelocity = CGPointMultiplyScalar(CGPointNormalize(desiredVelocity),self.
                  redSquare.maxSpeed);
59    }
60
61    return ( CGPointSubtract(desiredVelocity, self.redSquare.vectorVelocity));
62}
63
64
65-(CGPoint) pursuit:(Square *)evader {
66
67    CGPoint toEvader = CGPointSubtract(evader.position, self.redSquare.position);
68
69    double lookAheadTime = CGPointLength(toEvader)/(self.redSquare.maxSpeed + CGPointLength(
              evader.vectorVelocity));
70
71    CGPoint as = CGPointAdd(evader.position, CGPointMultiplyScalar(evader.vectorVelocity,
              lookAheadTime));
72
73    return [self seek:CGPointAdd(evader.position, CGPointMultiplyScalar(evader.
              vectorVelocity, lookAheadTime) )];
74}
75
76@end
```

APÊNDICE U – Estatísticas de Cobertura de Código

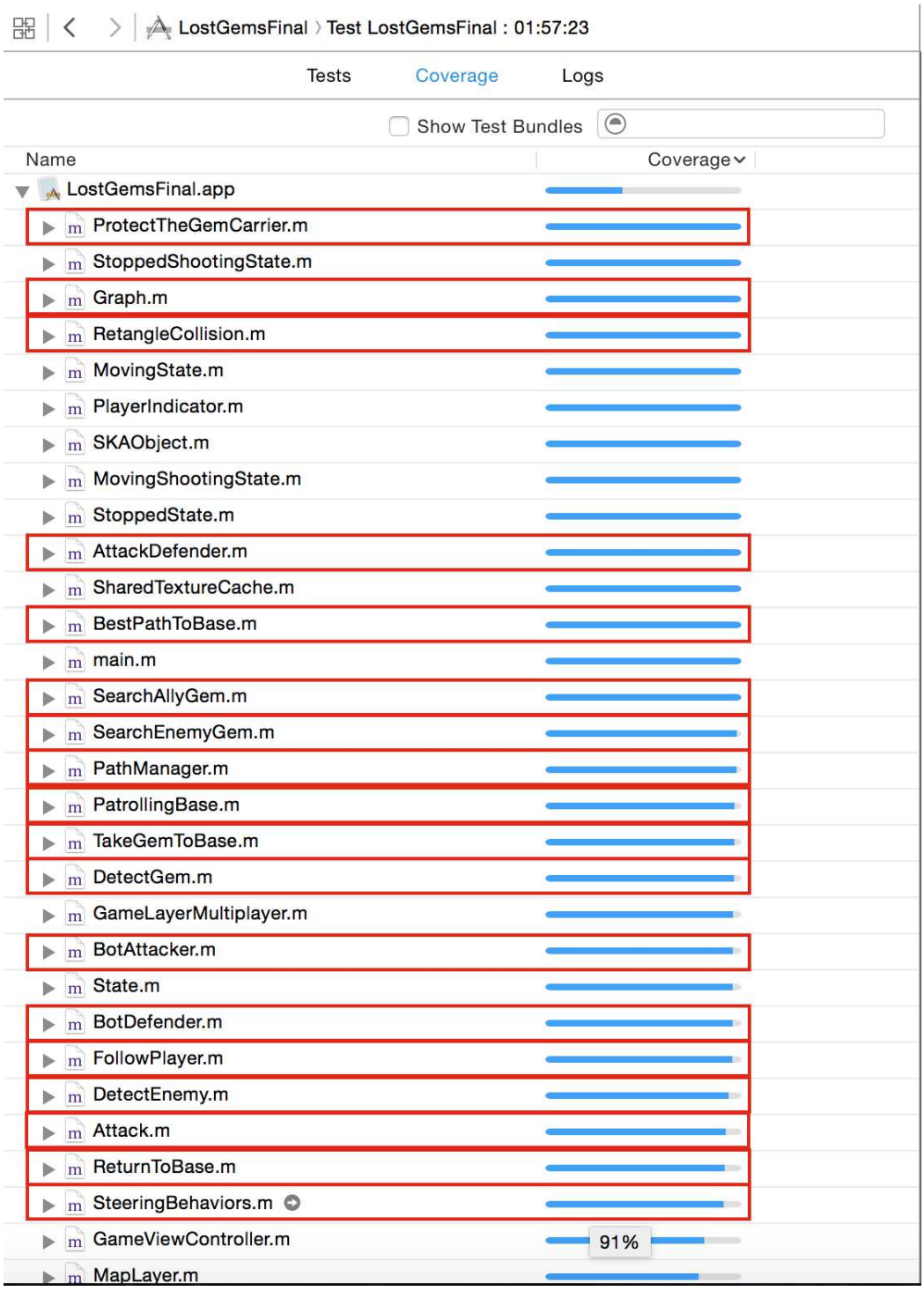

Figura 40: Estatísticas de Cobertura de Código

APÊNDICE V – Questionário de Avaliação dos *Bots*

QUESTIONÁRIO PARA AVALIAÇÃO DE AGENTES AUTOMATIZADOS PARA O JOGO LOST GEMS

1. Como você avalia a usa experiência com jogos eletrônicos?

Nada experiente	Pouco experiente	Razoável	Experiente	Muito experiente

2. De modo geral, como você avalia o comportamento dos *bots*?

Péssimo	Ruim	Razoável	Bom	Ótimo

3. Você teve muita dificuldade em derrotar os *bots* inimigos?

Não	Às vezes	Sim

3. Você se sentiu sozinho no jogo em relação aos *bots* aliados?

Não	Às vezes	Sim

4. O que achou da dificuldade do jogo?

Muito fácil	Fácil	Razoável	Difícil	Muito difícil

5. Como você avalia o nível de diversão e entretenimento da partida?

Muito tedioso	Tedioso	Razoável	Divertido	Muito divertido

6. É evidente que você está jogando com bots, sejam eles inimigos ou parceiros, e não com jogadores humanos?

Não	Às vezes	Sim

5. Quais sugestões/críticas você tem sobre o comportamento dos *bots*?

_____
_____
_____
_____

Figura 41: Questionário de Avaliação

APÊNDICE W – Documento de Reconhecimento

Eu, como co-criador e desenvolvedor do jogo *Lost Gems*, reconheço que todos os algoritmos que suportam a Inteligência Artificial dos *bots* para o referido jogo foram implementados por Bruno Rodrigues de Andrade no contexto das disciplinas Trabalho de Conclusão de Curso 1 e Trabalho de Conclusão de Curso 2.

1. _____
   Luís Rezende

2. _____
   Felipe Perius

3. _____
   Henrique Prudêncio

4. _____
   Thamara Gabriella

5. _____
   Bruno ROdrigues