FILIPPE DE CARVALHO LEITE
HENRIQUE DALL PIZZOL LAMPERT
THIAGO BEZERRA LOPES

INTELIGÊNCIA ARTIFICIAL ADAPTATIVA PARA UM JOGO DE TOWER DEFENSE

FILIPPE DE CARVALHO LEITE
HENRIQUE DALL PIZZOL LAMPERT
THIAGO BEZERRA LOPES

INTELIGÊNCIA ARTIFICIAL ADAPTATIVA PARA UM JOGO DE TOWER DEFENSE

Trabalho de conclusão de curso de Engenharia de Computação apresentado à Escola Politécnica da Universidade de São Paulo – Departamento de Engenharia de Computação e Sistemas Digitais

Orientador: Prof. Dr. Ricardo Nakamura

Catalogação-na-publicação

Leite, Filippe de Carvalho
Inteligência artificial Adaptativa para um Jogo de Tower Defense / F. C. Leite, T. B. Lopes, H. D. P. Lampert -- São Paulo, 2015.
46 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Inteligencia Artificial 2.Tower Defense 3.Adaptatividade I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Lopes, Thiago Bezerra IV.Lampert, Henrique Dall Pizzol

DEDICATÓRIA

À minha família pelo apoio incondicional desde quando me preparava para entrar na faculdade até o momento da conclusão deste trabalho. Aos meus amigos pelo apoio. À Bia Port pelo apoio, companheirismo e pelos puxões de orelha quando necessário.

Filippe de Carvalho Leite

Aos meus pais.

Henrique Dall Pizzol Lampert

Aos meus pais.

Thiago Bezerra Lopes

AGRADECIMENTOS

Agradecemos ao nosso professor orientador, Prof. Dr. Ricardo Nakamura, por ter aceitado orientar nosso projeto. Sempre que procurado, se prontificou a se reunir e fornecer a orientação necessária aos membros deste grupo para a conclusão do trabalho.

Agradecemos ao Prof. João José Neto pela orientação em uma conversa informal que esclareceu conceitos sobre inteligência artificial adaptativa e as possíveis maneiras de implementação no contexto deste trabalho.

RESUMO

Estudos de Inteligência Artificial para jogos digitais de estratégia raramente focam no nicho de jogos Tower Defense. Visando contribuir para essa área, este trabalho desenvolve e implementa algoritmos adaptativos para um jogo desse gênero. Além disso, desenvolve um jogo base capaz de testar os algoritmos implementados e mensurar as suas eficiências. Este jogo também pode servir de base para estudos futuros sobre o tema.

Palavras-chave: Tower Defense. Inteligência Artificial. Adaptabilidade. Algoritmo.

ABSTRACT

Studies about Artificial Intelligence applied to strategy games rarely focus on the Tower Defense niche. This paper aims to contribute to this area by developing and implementing adaptative algorithms for games on that genre. Besides that, develops a game capable to test the algorithms implemented and measure its efficiency. This game can also be the basis for future studies on the subject.

Keywords: Tower Defense. Artificial Intelligence. Adaptability. Algorithm.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Este capítulo apresenta a motivação, o objetivo, a justificativa e a estrutura da monografia.

1.1. Objetivo

Desenvolver uma inteligência artificial capaz de se adaptar a um jogo no estilo Tower Defense, defendendo-se das contínuas ondas de inimigos. A inteligência será baseada em algoritmos que modificam sua estratégia entre os turnos do jogo, controlando suas construções de acordo com as unidades enviadas, a disponibilidade de recursos e a atual situação do mapa. Ela escolherá as melhores posições para a construção de uma torre de defesa com o objetivo de diminuir a quantidade de sucessos dos inimigos.

Este trabalho também se propõe a elaborar um jogo base que permitirá o estudo de inteligências artificiais aplicadas em jogos de Tower Defense. O jogo proposto é amplamente customizável e dispõe de recursos para medir a eficiência da inteligência desenvolvida.

1.2. Motivação e Justificativa

A inteligência artificial é parte essencial em jogos competitivos que possuem modos para um ou múltiplos jogadores. Estes jogos se beneficiam da inteligência artificial implementando agentes que tomam decisões estratégicas baseadas em algoritmos pré-formulados. Tais algoritmos induzem os agentes a comportamentos determinísticos, baseados apenas em análises sobre o estado atual do jogo.

Este gênero de jogo, Tower Defense, não está normalmente associado a uma inteligência artificial complexa, a estratégia de defesa ocorre por parte do jogador humano, retira-lo e substitui-lo por uma inteligência artificial pode gerar comparações entre os dois. A utilização de algoritmos adaptativos impede que a inteligência implementada seja uma cópia do comportamento humano baseado em uma concepção do que se acha ser a melhor estratégia, a própria inteligência descobrirá sozinha as melhores decisões a serem tomadas.

A motivação para se utilizar inteligência artificial adaptativa é criar jogos mais imersivos nos quais os agentes devem se comportar de maneira inteligente, tomando decisões baseadas em estratégias elaboradas em resposta às interações do jogo ao longo de toda partida. O estudo de técnicas adaptativas neste contexto também pode levar ao aperfeiçoamento da concepção do desenvolvimento de jogos Tower Defense.

1.3. Estrutura da Monografia

O documento é organizado em 7 capítulos: Introdução, Conceitos e Tecnologia, Especificação do Jogo Base, Especificação da Inteligência, Desenvolvimento, Resultados e Conclusão.

No capítulo 1 - Introdução – são apresentadas as seções de Objetivo, Motivação e Justificativa e a Estrutura da Monografia.

No capítulo 2 - Conceitos e Tecnologias - são apresentadas as seções de tecnologias utilizadas e suas justificativas, assim como os conceitos empregados, abrangendo: Inteligência Artificial, Adaptabilidade, Tower Defense, libGDX, Java, Git e Bitbucket.

No capítulo 3 - Especificação do Jogo Base – são apresentadas as seções: Descrição Geral, Funcionamento, Visão Geral, Estrutura do Jogo, Elementos Visuais e Logs e Feedbacks Visuais.

No capítulo 4 - Especificação da Inteligência - são apresentadas as seções: Descrição Geral, Algoritmo de Pesos e Algoritmo de Ordenação.

No capítulo 5 - Desenvolvimento - são apresentadas as seções Resumo das Atividades e Gerenciamento do Projeto

No capítulo 6 - Resultados – são apresentados as seções: Distribuição Aleatória, Algoritmo de Pesos, Algoritmo de Ordenação e Comparações entre Algoritmos.

No capítulo 7 - Conclusão – são apresentadas as seções: Considerações Finais, Contribuição e Trabalhos Futuros.

2. CONCEITOS E TECNOLOGIA

Este capítulo apresenta os conceitos e definições de Inteligência Artificial, Adaptabilidade e do gênero de jogo Tower Defense, também apresenta as tecnologias libGDX, Java, Git, BitBucket que foram utilizadas neste projeto.

2.1. Inteligência Artificial

A inteligência artificial (IA) é a ciência e a engenharia de produzir máquinas inteligentes, em especial programas de computador inteligentes. Está relacionada à tarefa de usar computadores para entender a inteligência humana, porém, uma IA não se limita a métodos biológicos observáveis (MCCARTHY, 2012).

Possui diversas aplicações hoje em dia como o reconhecimento de fala, o entendimento da linguagem natural, a visão computacional, os sistemas especialistas e, como abordado neste documento, a interação com jogos eletrônicos. Desde o nascimento dos jogos digitais, a IA tem sido um elemento de estudo para o desenvolvimento destes jogos. Ela abrange diversas áreas como a interação, o machine learning, a escala de dificuldade e a tomada de decisões em um jogo. Uma IA sofisticada pode aumentar o entretenimento fornecido por um jogo e, consequentemente, a receita ganha por ele.

A adaptação de um jogo ao jogador deve ser direcionada por um propósito que possa ser identificado, medido e influenciado pelo desenvolvedor do jogo. A partir disto, decisões podem ser tomadas determinando como e quando o jogo irá reagir de acordo com gatilhos específicos.

Todos os componentes de um jogo podem ser adaptativos e podem contribuir para a experiência do jogador, a mecânica do jogo, ambientes, cenários, NPCs (non playing characters), narrativas e desafios. Essa adaptação pode ser off-line, dependendo dos dados do jogador quando este inicia o jogo ou um novo desafio, geralmente associado ao carregamento de uma nova fase, ou on-line, o que neste caso significa em tempo real, ou seja, enquanto o jogador está jogando, seus dados são monitorados e comparados com o modelo inicial do jogador, isto pode acarretar em uma nova adaptação ou atualização do modelo do jogador. (ROHTKRANTZ; KOPPELAAR; SPRONCK, 2004)

2.2. Tower Defense

2.2.1. Definição

Tower Defense (TD) é um gênero de jogo de estratégia tipicamente com fundamentos simples. O jogador, normalmente humano, coloca torres para defender um caminho pré-definido que é percorrido por ondas de inimigos. O jogador perde se não conseguir impedir que um número máximo de inimigos cheguem ao final do caminho. Este gênero é caracterizado pelo posicionamento fixo de unidades de defesa (torres) contra unidades inimigas móveis. Apesar de possuir uma definição simples, jogos de Tower Defense podem se diferenciar bastante em sua jogabilidade e composição, tornando o gênero muito rico em variações. (RUMMEL, 2011)

A Figura 1 representa um típico jogo de Tower Defense, os inimigos percorrem o caminho que sai do canto superior da imagem com o objetivo de chegar até as duas bandeiras no lado direito da imagem. As torres de defesa estão espalhadas pela área verde e não podem ser construídas no caminho das unidades inimigas. É possível também visualizar no canto superior esquerdo o número máximo de inimigos que podem alcançar seu objetivo e a quantidade atual de recursos. Próximo ao centro da imagem existe uma torre prestes a ser construída, neste caso o jogador pode escolher entre 4 tipos diferentes de torres e a quantidade de recursos gastos para sua construção.

Figura 1 – Screenshot do jogo Kindom Rush. Fonte: Site do jogo Kingdom Rush.

2.2.2. Breve História

O primeiro jogo comercial que pode ser considerado um Tower Defense chama-se Rampart, para a plataforma Atari em 1990. Suas similaridades com os TDs atuais se devem ao princípio do jogo ser defender um território a partir da criação de unidades que deveriam ser reparadas entre rodadas de ataques.

A expansão The Frozen Throne do jogo Warcraft III, produzido pela Blizzard em 2003, foi a primeira a reunir elementos dos Tower Defense atuais da maneira como conhecemos. Os jogadores eram capazes de criar seus próprios mapas e competir entre si quem destruía mais unidades inimigas. (AVERY et al., 2011)

Atualmente estes jogos são muito populares em versões em Flash e para dispositivos móveis, possuindo variações em todas as áreas do jogo.

2.2.3. Composição

Este gênero geralmente é composto de um mapa, torres de defesa, unidades inimigas e recursos. Os elementos básicos de um *Tower Defense* são definidos a seguir.

2.2.3.1. Mapa

O mapa é o terreno por onde as peças inimigas irão transitar e também onde serão colocas as torres de defesa. Na maioria destes jogos, as unidades inimigas seguem um caminho fixo de um ponto de entrada no cenário até o destino, geralmente a base do jogador.

2.2.3.2. Torres de Defesa

As torres são unidades fixas colocadas pelo jogador no mapa. Possuem a função de atacar as hordas de unidades inimigas. Dependendo do jogo, existem tipos de torres com função secundária, como recuperar energia de torres vizinhas ou atrasar o avanço de unidades adversárias.

2.2.3.3. Unidades Inimigas

Estas unidades podem ser ou não capazes de atacar as torres e devem ser impedidas de percorrerem todo o caminho do mapa. É comum que existam tipos diferentes, com características diferentes como velocidade, ataque, defesa e fraquezas. Unidades mais lentas costumam ser mais difíceis de se destruir do que as mais rápidas. Também é comum na maioria destes jogos uma unidade inimiga ‘chefe’ que é mais difícil de se destruir e normalmente enviada ao final de um nível.

2.2.3.4. Recursos

Os recursos estão presentes na maioria dos Tower Defense e são utilizados para limitar a construção de novas torres no mapa, com cada tipo de torre possuindo um custo.

Além disto, recursos também podem ser utilizados para aperfeiçoar as características das torres. Alguma quantidade de recursos geralmente é dada ao jogador como recompensa a cada unidade inimiga destruída ou onda concluída.

2.3. libGDX

O libGDX é um framework de desenvolvimento multi-plataforma para jogos que provê um ambiente para rápidas prototipações e iterações. Escolhemos este framework pelo grupo já possuir familiaridade com o libGDX e por sua facilidade de desenvolvimento para diversas plataformas (Windows, Linux, Mac OS X, Android, BlackBerry, iOS e HTML/WebGL) sem a necessidade de modificações. Sua comunidade é ampla e consolidada, o que favorece o suporte a dúvidas. A linguagem utilizada para desenvolvimento é Java. (LIBGDX, 2013)

2.4. Java

Java é uma linguagem de programação orientada a objetos lançada pela Sun em 1995. Este time era liderado por James Gosling, considerado hoje o pai desta linguagem. Diferentemente de linguagens convencionais, utiliza uma máquina virtual, entre o sistema operacional e a aplicação, que é responsável por traduzir e fazer as chamadas para o sistema operacional, desta forma não é necessário se preocupar com as especificações do sistema no qual sua aplicação está rodando. (CAELLUM, 2015)

O Java, atualmente adquirido pela Oracle, se mantém como a linguagem de programação mais popular, à frente das linguagens C, C++ e Python. Todos os integrantes do grupo já desenvolveram aplicações em Java, isso facilitou a elaboração deste projeto e favoreceu para que todos pudessem colaborar. (CASS, 2015)

2.5. Git

O Git é um sistema de controle de versão distribuído, ou seja, os usuários não fazem somente cópias das últimas versões dos arquivos, e sim cópias completas do repositório. Desta maneira, se alguma falha acontece, os repositórios podem ser

restaurados a partir das cópias dos usuários. Estes sistemas são ótimos para ambientes colaborativos, permitindo o trabalho em conjunto com outras pessoas simultaneamente no mesmo projeto. O Git é utilizado em desenvolvimento de software com ênfase em velocidade e integridade de dados. Foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel do Linux em 2005. (CHACON; STRAUB, 2014)

Escolhemos esse sistema por ser amplamente utilizado atualmente para desenvolvimento de software e por todos os integrantes do grupo já o terem utilizado como controle de versão.

2.6. Bitbucket

O Bitbucket é um serviço de hospedagem para projetos que utilizam Mercurial ou Git como sistema de controle de versão. Escolhemos o Bitbucket por possuir repositórios privados gratuitos para equipes de desenvolvimento de até 5 usuários, diferentemente do GitHub, a opção mais conhecida e comum para desenvolvedores que queiram usar o Git como sistema de versionamento, que só possui repositórios privados pagos. Outro motivo de termos optado por esse serviço é a familiaridade que temos com o Git e com o próprio Bitbucket.

2.7. Considerações Finais do Capítulo

Os conceitos apresentados neste capítulo definem as tecnologias utilizadas no jogo e nos algoritmos projetados, assim como suas respectivas arquiteturas. Para a escolha das tecnologias foram ponderados os seguintes quesitos:

- Curva de aprendizagem e familiaridade;
- Relevância da tecnologia no mercado;
- Capacidade de reutilização dos produtos do desenvolvimento.

Portanto, as tecnologias escolhidas para o desenvolvimento:

- LibGDX como framework de desenvolvimento para o jogo;

- Java como linguagem de programação, utilizando a IDE IntelliJ IDEA 15.0;
- Git como sistema de controle de versão, utilizando o BitBucket como serviço de hospedagem do projeto.

3. ESPECIFICAÇÃO DO JOGO BASE

3.1. Descrição Geral

O jogo base proposto por este documento foi inicialmente desenvolvido para atender as necessidades da inteligência projetada para este trabalho e testar seu desempenho. Ao longo do processo, o grupo percebeu a possibilidade de torná-lo uma ferramenta para estudos sobre Inteligência Artificial aplicada em jogos de Tower Defense.

O produto deste desenvolvimento foi um jogo altamente customizável e expansível que possui uma arquitetura simples de ser compreendida. É possível incluir diferentes unidades, novas inteligências artificias, novas estratégias de envio de ondas inimigas ou até mesmo modificar as suas regras sem dificuldades.

Além disso, para que seja possível testar e medir o desempenho de inteligências implementadas, foram incluídos logs e feedbacks visuais sobre diferentes dados fornecidos pelo jogo. Estes logs também são customizáveis, é possível acrescentar outros medidores de desempenho que não foram abrangidos por este trabalho.

Espera-se que este jogo base sirva como uma ferramenta para colaborar com outros estudos sobre o tema.

3.2. Funcionamento

O jogo é regido por um mecanismo de turnos. A cada turno, uma nova onda de inimigos é enviada. Antes disso, a inteligência artificial deve decidir onde construir suas novas torres. As torres construídas em turnos anteriores permanecem no mapa. O jogador começa com uma quantidade limitada de recursos e ganha uma certa quantidade a mais de recursos a cada inimigo abatido. A construção de uma

nova torre possui um custo, que deve ser abatido da quantidade de recursos do jogador. Esse mecanismo visa limitar o número de torres que o jogador é capaz de construir a cada novo turno. Para os testes realizados nesse trabalho, o jogo base foi configurado com os seguintes mecanismos de funcionamento e parâmetros:

- Apenas um tipo de torre, capaz de causar um ponto de dano;
- Cada torre tem um custo de construção igual a 10;
- Apenas um tipo de inimigo, variando apenas a quantidade total de dano que ele é capaz de receber;
- O jogador começa com 100 pontos de recursos e ganha 1 ponto a cada inimigo abatido;
- A onda inicial tem 30 inimigos. A cada 5 ondas, o número de inimigos aumenta em 5. O número de ondas é infinito;
- Os inimigos são capazes de receber um dano igual ao número da onda a qual pertencem. Isto é, um inimigo da onda 6 precisa receber 6 pontos de dano para morrer;
- Se 50 inimigos chegarem ao final do caminho, o jogador perde o jogo. Em outras palavras, o jogador tem 50 pontos de vida, perde um desses pontos a cada inimigo que alcança o final do caminho, e perde o jogo se esse número chegar a zero.

O objetivo é que os algoritmos desenvolvidos sejam capazes de aprender a jogar, melhorando seu desempenho a cada nova tentativa de vencer o jogo. A seguir, é mostrado um fluxograma de uma sessão típica de jogo. Nele, uma nova sessão representa uma nova tentativa inicial de jogo. Nesse passo, os algoritmos voltam ao seu estado inicial. Um novo jogo é uma nova tentativa, o jogo volta para o estado inicial e as ondas são enviadas novamente desde a primeira. Uma nova onda representa o começo de um novo turno: o jogador constrói suas torres e uma nova onda é enviada. O jogo se encerra quando o jogador não tiver mais pontos de vida ou não existirem mais ondas a enviar.

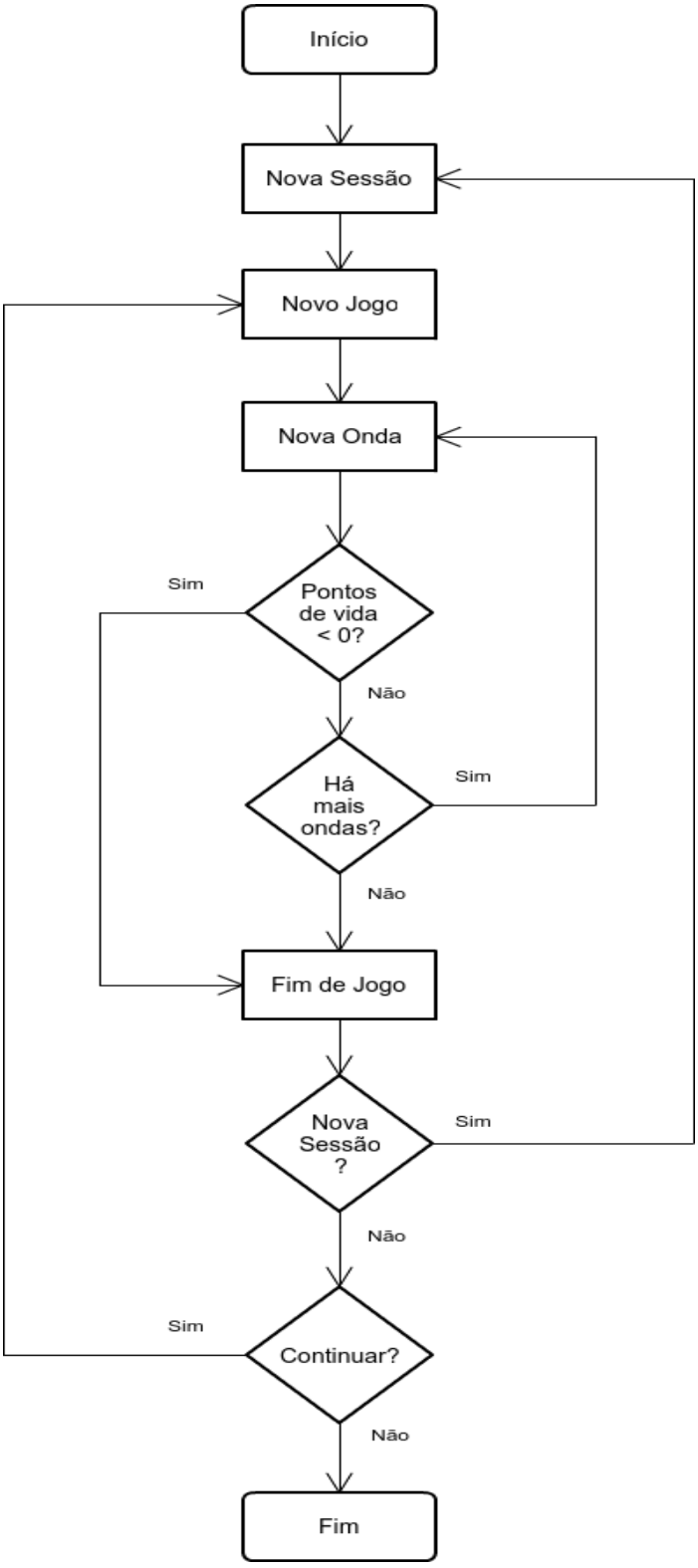


Figura 2 – Fluxograma do jogo base

3.3. Visão Geral

Apesar de parte essencial em jogos de estratégia, a Inteligência Artificial é apenas um entre outros módulos do sistema que constitui um jogo digital, como se pode notar no diagrama WBS (Working Breakdown Struct) abaixo:

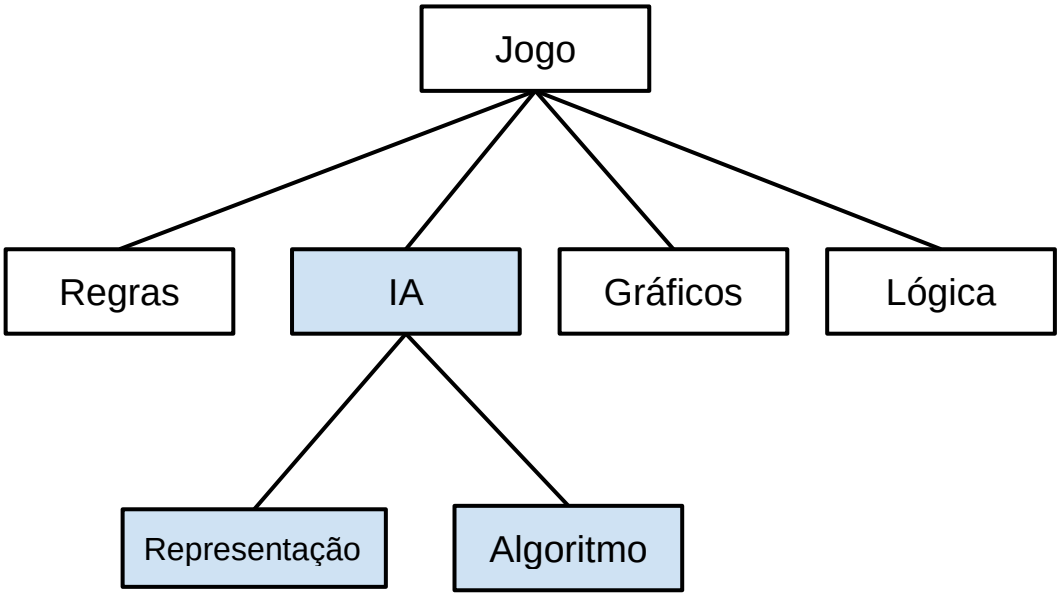


Figura 3 – Diagrama WBS. Fonte: Elaborado pelos autores.

3.4. Estrutura do Jogo

O jogo base foi dividido em quatro componentes principais: o **controlador do jogo**, responsável pelo controle da lógica do jogo e pela integração de todos os componentes; a mecânica de carregamento e manipulação de informações referentes ao **mapa**; o mecanismo de envio de inimigos e **controle de ondas**; e a própria **inteligência artificial**, que assume o papel de jogador e tem como objetivo impedir que as hordas inimigas cheguem ao final do caminho.

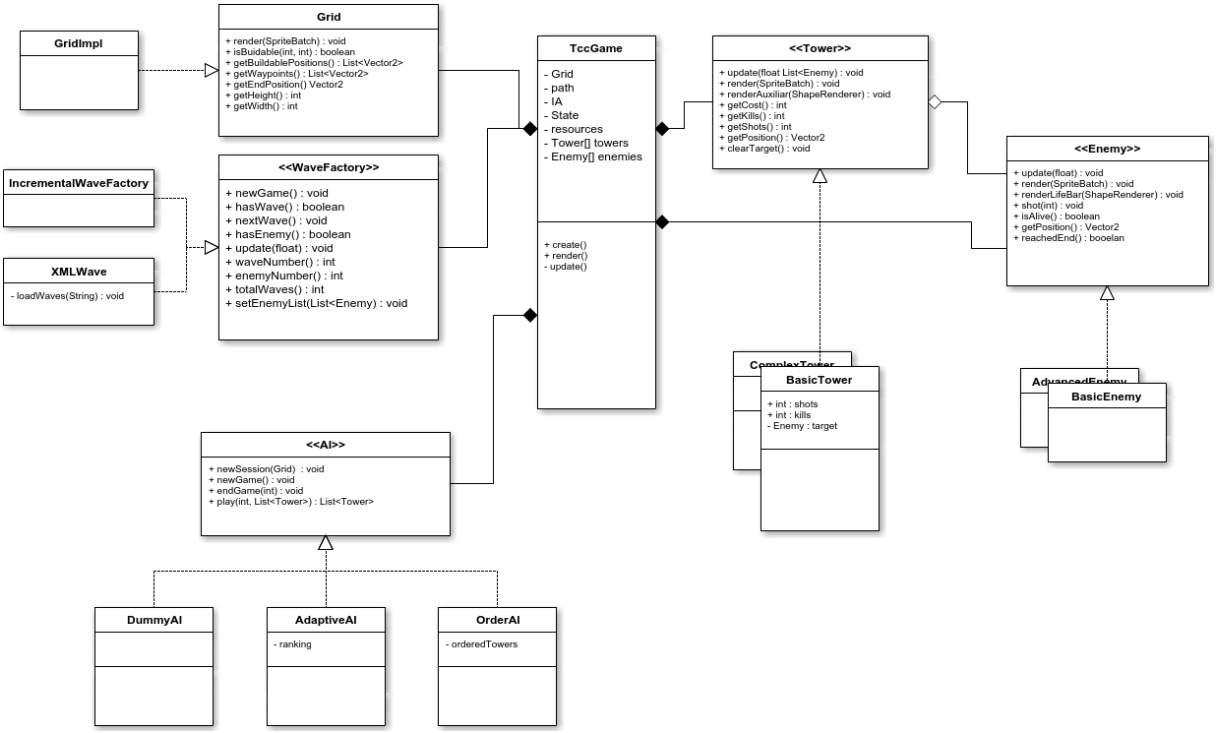


Figura 4 – *Diagrama de classes do jogo base.* Fonte: Elaborado pelos autores

3.4.1. Controlador

O módulo de controle, representado pela classe TccGame no diagrama acima, mantém uma instância de cada um dos outros componentes principais do jogo.

A comunicação com cada componente é feita através de uma interface padronizada, permitindo a substituição de cada um deles, facilitando a modificação do funcionamento do jogo. Quando uma nova sessão é iniciada, o controlador instancia cada um dos outros módulos. A cada passo de atualização do jogo, o controlador atualiza o estado de cada inimigo e de cada torre, verifica se algum inimigo alcançou o final do caminho e se comunica com o *WaveFactory* instanciado para saber se existem ainda mais ondas ou mais inimigos a serem enviados. De acordo com essas informações, o controlador muda seu estado interno e toma as ações necessárias. Seu funcionamento é baseado em uma máquina de estados que modela o fluxograma do jogo, mostrado na figura 2.

3.4.2. Mapa

Modelado na classe Grid do diagrama mostrado na figura 4. O mapa de jogo permanece o mesmo a cada sessão. As responsabilidades do Grid são carregar o mapa e informar quais são as posições livres para a construção de torres e qual o caminho percorrido pelos inimigos.

Grid:

- render(SpriteBatch): void - deve renderizar o mapa.
- isBuildable(int, int): boolean - retorna se a coordenada passada corresponde a uma posição válida para a construção de uma torre - se não está no caminho dos inimigos.
- getBuildablePositions(): List<Vector2> - retorna uma lista com todas as posições do mapa onde é possível construir uma torre.
- getWaypoints(): List<Vector2> - retorna uma lista com os pontos de rota do caminho dos inimigos.
- getEndPosition(): Vector2 - retorna as coordenadas da última posição do caminho dos inimigos.
- getHeight(): int - o número de células o mapa possui de altura.
- getWidth(): int - o número de células que o mapa tem de largura.

3.4.3. Controle de Ondas

O mecanismo de controle de ondas foi modelado na interface WaveFactory. O que se espera de um controlador de ondas é que ele gerencie o envio de ondas e de inimigos. É dele a responsabilidade de carregar novas ondas quando solicitado pelo controlador e de instanciar e enviar novos inimigos para o controlador no momento adequado, de acordo com alguma lógica interna.

WaveFactory:

- newGame(): void - reinicializa a *factory.*
- hasWave(): boolean - informa se há ou não ondas a enviar.
- nextWave(): void - informa à *factory* que uma nova onda deve ser iniciada.

- hasEnemy(): boolean - retorna se a onda atual possui ou não mais inimigos a enviar.
- update(float): void - chamado a cada passo de atualização. É a partir de chamadas a esse método que a *factory* deve decidir quando instanciar novos inimigos.
- waveNumber(): int - retorna o número da onda atual.
- enemyNumber(): int - retorna o número total de inimigos da onda atual.
- totalWaves(): int - retorna o número total de ondas.
- setEnemyList(List<Enemy): void - lista onde a *factory* deve colocar os inimigos que deseja criar.

3.4.4. Inteligência Artificial

A comunicação do módulo de inteligência artificial com o controlador é feita através da interface AI. A inteligência é informada da mudança de estado do jogo por meio de alguns métodos, enquanto o método *play* deve servir como saída para a jogada dela a cada nova onda. O controlador espera receber como jogada uma lista de torres - as torres que o jogador deseja construir no turno atual.

AI:

- newSession(Grid): void - é chamado quando uma nova sessão do jogo é iniciada. A IA deve voltar para o seu estado inicial.
- newGame(): void - chamado quando uma nova sequência de ondas é iniciada. Corresponde a uma nova tentativa de vencer o jogo.
- endGame(int): void - o final do jogo foi alcançado. Ou o jogador perdeu ou não existem mais ondas a enviar. Recebe a quantidade de mortes alcançada nessa tentativa.
- play(int, List<Tower>): List<Tower> - é o início de um novo turno. O jogador deve informar quais torres deseja construir de acordo com os recursos de que dispõe e com as torres já construídas.

Conforme mostra o diagrama da figura 4, além dos módulos relacionados ao funcionamento do jogo, foram criados dois grupos de classes: torres e inimigos. O comportamento de torres e inimigos foram modelados em interfaces independentes para tornar a adição de novos tipos de torres e inimigos fácil de implementar e completamente independente das outras partes do jogo.

Tower:

- update(float, List<Enemy>): void - a torre deve atualizar seu estado. Recebe o tempo decorrido desde a última atualização e o número de inimigos presentes no mapa no instante atual.
- render(SpriteBatch): void - a torre deve renderizar a si mesma.
- renderAuxiliar(ShapeRenderer): void - chamada para a torre renderizar informações secundárias sobre seu estado, como raio de ação e linha de tiro.
- getCost(): int - retorna o custo da torre.
- getKills(): int - retorna o número total de inimigos abatidos pela torre.
- getShots(): int - retorna o número total de disparos efetuados pela torre.
- getPosition(): Vector2 - retorna a posição da torre no mapa.
- clearTarget(): void - a torre deve liberar qualquer alvo que tenha mantido.

Enemy:

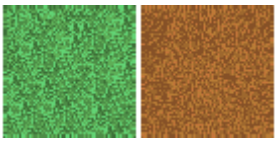
- update(float): void - o inimigo deve atualizar seu estado. Recebe o tempo decorrido desde a última atualização.
- render(SpriteBatch): void - o inimigo deve cuidar da própria renderização.
- renderLifeBar(ShapeRenderer): void - solicita que o inimigo renderize sua barra de vida.
- shot(int): void - informa que o inimigo foi atingido. Recebe a quantidade de dano causado.
- isAlive(): boolean - retorna se o inimigo está ou não vivo.
- getPosition(): Vector2 - retorna a posição que o inimigo ocupa no mapa.

- reachedEnd(): boolean - retorna se o inimigo alcançou ou não o final do caminho.

3.5. Elementos visuais

As imagens utilizadas no jogo são carregadas a partir do arquivo texture.png e podem ser modificadas ou expandidas. A configuração inicial dos elementos visuais do jogo base assim como sua customização são descritas a seguir.

A textura utilizada para o terreno onde as torres são colocadas possuem uma coloração verde semelhante a grama, já a textura do caminho percorrido possui uma coloração marrom semelhante a terra.



Figuras 5 e 6 – Texturas utilizadas para ilustrar o terreno e caminho respectivamente.

As unidades inimigas e as torres de defesas são apresentadas a seguir, a primeira ilustra um zumbi, a segunda um soldado carregando uma arma, ambas vistas de cima.

Figuras 7 e 8 – Imagens utilizadas para ilustrar respectivamente unidades inimigas e torres de defesa.

O mapa é descrito por um arquivo de texto (map.txt) contendo somente números como visto na imagem, o número 2 representa o início e 3 representa o fim do caminho percorrido pelas unidades inimigas, já o número 0 representa todas as possíveis posições em que torres de defesa podem ser construídas.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0
3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figura 9 – Arquivo texto utilizado para geração do mapa. Fonte: Elaborado pelos autores.

A figura 10 representa o mapa após renderização tendo como entrada o arquivo de texto da figura 9.

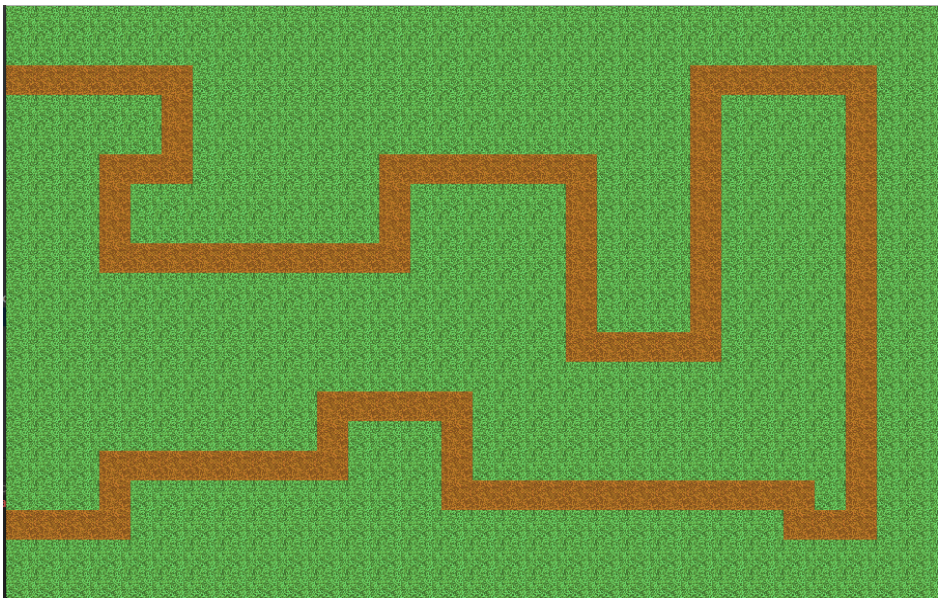


Figura 10 – Resultado do mapa após renderização. Fonte: Elaborado pelos autores

3.6. Logs e Feedbacks Visuais

3.6.1. Log

O jogo imprime no console da IDE e em um arquivo de texto as informações mais relevantes sobre cada partida jogada pela IA escolhida para jogar. A saída é atualizada ao final de cada nova onda de inimigos, de maneira que o log sempre mostra as informações mais atualizadas a respeito do desempenho da IA e sobre o estado atual do jogo.

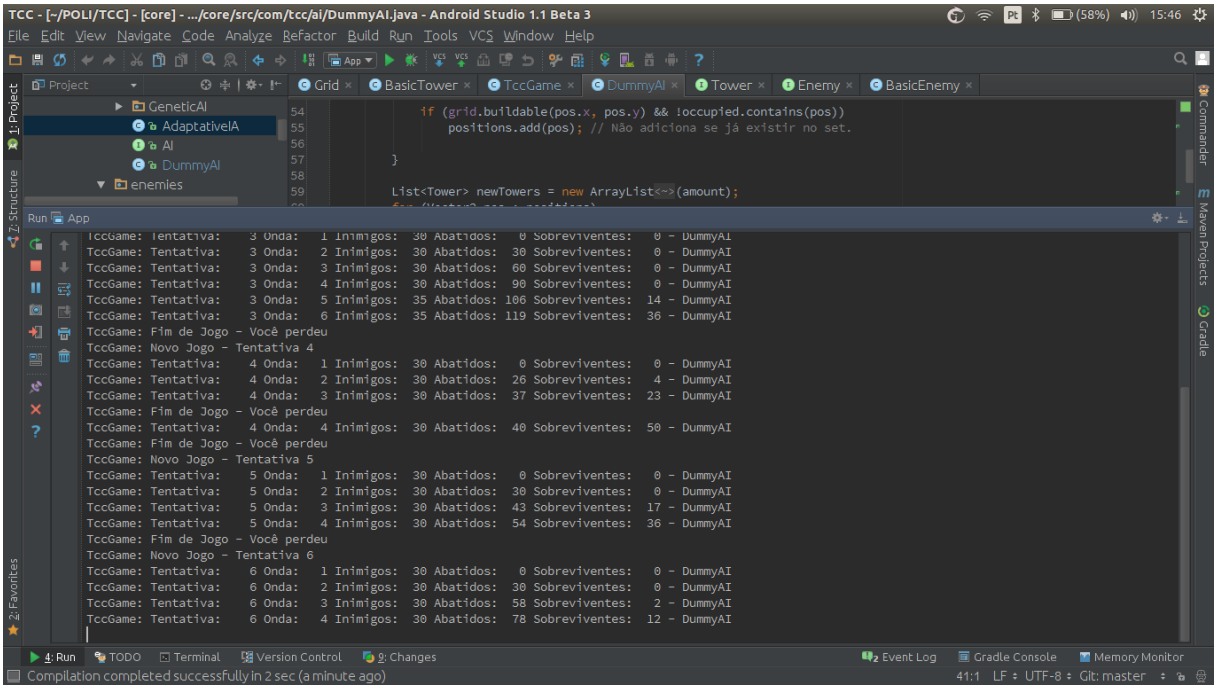


Figura 11 - Log impresso no console. Fonte: Printscreen da IDE Intellij IDEA 15

As informações apresentadas no log abrangem os seguintes dados:

- Tentativa: indica o número de vezes que a IA já reiniciou o jogo. O reinício acontece quando um número mínimo de inimigos chega ao final do mapa;
- Onda: é o número da sequência atual de inimigos. Este valor pode ser entendido também como o número de jogadas a IA realizou ao longo do jogo em andamento;
- Inimigos: informa quantos inimigos foram enviados na última onda;

- Abatidos: quantos dos inimigos enviados na última onda foram derrotados antes de chegarem ao final do mapa;
- Sobreviventes: quantos dos inimigos enviados na última onda chegaram vivos ao final do mapa.
- O identificador da IA que está jogando.

3.6.2. Feedbacks Visuais

O jogo também pode imprimir na tela os pesos atribuídos pela inteligência artificial às diferentes células do mapa.

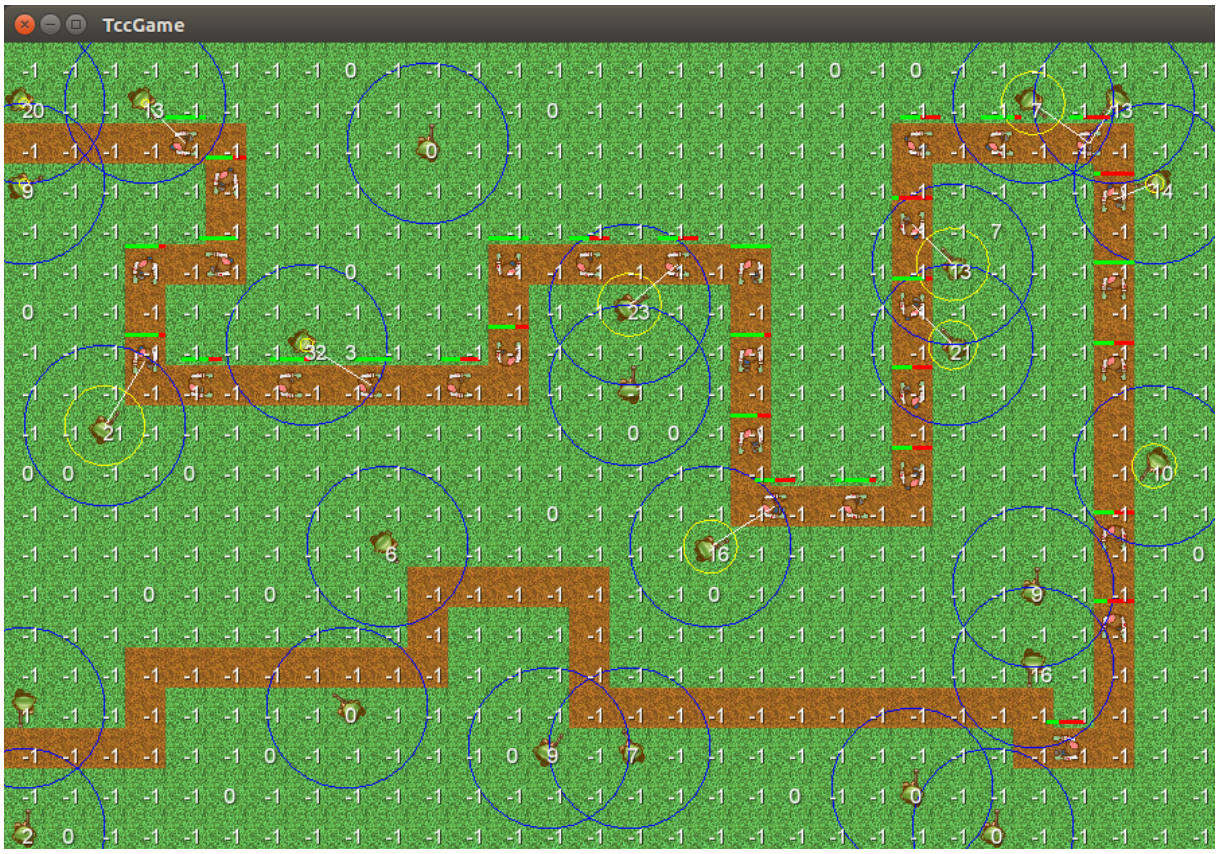


Figura 12 - Pesos impressos na tela do jogo. Fonte: Elaborado pelos autores

4. ESPECIFICAÇÃO DA INTELIGÊNCIA

4.1. Descrição Geral

Este projeto se propõe a elaborar uma inteligência artificial capaz de jogar um jogo de Tower Defense, que de acordo com os dados coletados do próprio jogo, é capaz de descobrir as melhores posições para a construção de torres e, ao mesmo tempo, administrar recursos.

Os dois algoritmos de inteligência artificial desenvolvidos para este projeto serão explicados abaixo.

4.2. Algoritmo de Pesos

4.2.1. Descrição do algoritmo

Este algoritmo se baseia em pesos atribuídos a cada posição do mapa, atualizados a cada turno, para decidir onde colocar as torres. Na primeira sessão de jogo esses pesos, ou também *rankings*, são todos inicializados com o valor -1, que representa na lógica do algoritmo uma posição inexplorada, isto é, que não recebeu nenhuma torre ainda.

A cada novo turno, a IA começa atualizando os valores dos *rankings* de acordo com a regra estipulada. Esta regra pode considerar diversas estatísticas do jogo, como, por exemplo, a quantidade de inimigos mortos por uma torre específica, ou o número da onda atual. Para o algoritmo implementado, a regra é atribuir para cada ranking com posição ocupada o valor do número total de disparos dentro do turno da torre posicionada naquela coordenada.

Após atualizar os *rankings*, o algoritmo reserva uma porcentagem dos recursos disponíveis para posteriormente utilizar na construção de torres em lugares não visitados. Os recursos que sobram são usados na construção de torres seguindo a ordem dos pesos, do maior valor até o menor. Caso só haja posições inexploradas no mapa, o algoritmo escolhe lugares aleatórios, e que não façam parte do caminho, para criação de torres até que se esgotem os recursos daquele turno.

4.2.2. Pseudocódigo

```
inicialização:
para cada posição p do mapa
      ranking[p] ← -1
embaralha lista de posições

iteração:
para cada posição p = ocupada
      ranking[p] = calcula_ranking(p)

reserva ← teto(recursos * porcentagem_para_reserva)
recursos ← recursos - reserva

enquanto houver posição p = livre e p != path
                              e reserva >= custo_torre
      mapa[p] ← nova torre
reserva = reserva - custo_torre

ordene lista de posições segundo ranking
para todo p em ranking
      se p = livre e p != path e recursos >= custo_torre
            mapa[p] ← nova torre
            recursos = recursos - custo_torre
```

4.3. Algoritmo de Ordenação

4.3.1. Descrição do algoritmo

A ideia principal desse algoritmo é montar uma fila de construções. Ele começa com uma lista embaralhada de torres, uma em cada posição válida do mapa. As torres são construídas de acordo com a ordem em que aparecem na lista. Ao final de cada onda, os disparos de cada torre são contabilizados, e as torres construídas são reordenadas na lista de construções. As torres que mais dispararam são movidas para posições mais à frente, enquanto aquelas que menos dispararam são movidas para trás. Ao final de cada tentativa de vencer o jogo, a lista é varrida até a última posição ocupada por uma torre que tenha sido construída. Nessa varredura, as torres que não tiverem disparado são movidas para as últimas

posições da lista. Isso faz com que essas torres só venham a ser construídas em último caso, quando não existirem mais posições não testadas.

Diferente dos outros algoritmos apresentados nesse trabalho, o algoritmo de ordenação só usa aleatoriedade quando não sabe que posição escolher. A exploração de áreas desconhecidas só acontece após ele colocar todas as torres úteis que ele conhece na tentativa atual.

4.3.2. Pseudocódigo

```
inicialização:
      embaralhe lista de posições do mapa

iteração:

      índice ←0

      para cada posição p em que mapa[p] = ocupado
            se torre[p] não disparou
                  envia p para final da lista de posições
            se não
                  índice ←índice + 1

            ordena posições[0…índice] segundo quantidade
            de disparos de cada torre com posição em posições[0...índice]

            para todo p em posições
                  se p = livre e p != path e recursos >= custo_torre
                        mapa[p] ← nova torre
                        recursos = recursos - custo_torre
```

5. DESENVOLVIMENTO

Este capítulo apresenta um Resumo das Atividades e o Gerenciamento do projeto.

5.1. Resumo das Atividades

No início da primeira disciplina, Projeto de Formatura I, o grupo teve dificuldades em conseguir um tema e um orientador que estivessem com as expectativas alinhadas com o grupo. A proposta de um projeto na área de jogos eletrônicos surgiu por interesse e uma familiaridade prévia com algumas tecnologias de desenvolvimento de jogos. A partir disso, a escolha óbvia para orientação foi o Prof. Dr. Ricardo Nakamura, docente responsável pela disciplina de Design e Programação de Games, optativa para o curso de Engenharia de Computação.

Inicialmente, o foco deste projeto era somente a elaboração de uma inteligência artificial para jogos de Tower Defense, e para isso um kit de desenvolvimento do motor de jogo Unity seria utilizado. Devido a falta de familiaridade com a tecnologia e o tempo escasso, foi decido que seria produzido um jogo base simples que fosse capaz de fornecer os dados necessários para mensurar a eficiência dos algoritmos especificados. Este jogo seria desenvolvido utilizando o framework libGDX.

O projeto foi dividido em etapas, cada qual com uma inteligência cada vez mais complexa. Isto permitiu que, independentemente do tempo escasso, ao fim de cada etapa o grupo teria um produto funcionando capaz de ser apresentado.

5.2. Gerenciamento do Projeto

O processo de gerenciamento do projeto teve como inspiração o SCRUM, que é um processo de desenvolvimento de software ágil com características incrementais e iterativas. (VIEIRA, 2004)

O sprint é uma atividade básica do SCRUM, ela possui uma duração pré-definida e é quando as funcionalidades e outras etapas do produto final são de fato desenvolvidas. Antes de cada sprint são definidas as funcionalidades do projeto a serem elaboradas durante o próximo sprint a partir de um backlog, conjunto de todas

as funcionalidades do produto. Utilizamos as datas de entrega dos relatórios parciais deste projeto como datas finais dos sprints. Cada planejamento de sprint era decidido com os 3 integrantes do grupo após uma reunião com o orientador para expor o progresso das atividades e para que fossem dadas as orientações pra a próxima etapa.

A distribuição do tempo de dedicação ao trabalho foi afetada por provas e trabalhos durante o quadrimestre. Estas dificuldades foram contornadas retirando algumas das funcionalidades propostas no sprint e adicionando a um próximo. A figura 13 apresenta o backlog, assim como o progresso das funcionalidades e atividades envolvidas neste projeto.

Como pode se verificar na figura 13, as funcionalidades passam por um processo de revisão, este processo é denominado *code review* e consiste em uma revisão do código feita por um dos membros do grupo que não seja o próprio autor do código. Para que o código seja adicionado à estrutura principal do jogo, é necessário que ele seja revisto e aprovado por um dos membros do grupo.

Foi utilizado o Trello, uma aplicação de gerenciamento de projetos gratuita, para que fosse possível organizar o projeto e os sprints.

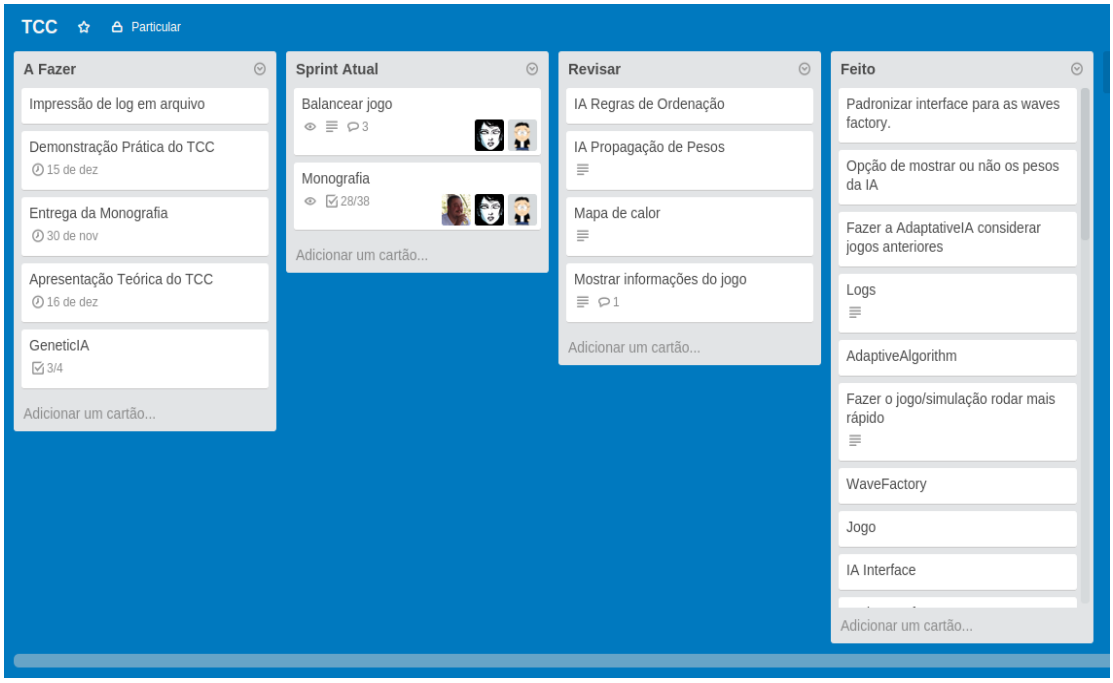


Figura 13 - Backlog e andamento das atividades. Fonte: Printscreen do site Trello.

6. RESULTADOS

6.1. Distribuição Aleatória

Como mecanismo de controle, foi implementada uma inteligência artificial burra. A cada jogada, ela apenas constrói o máximo de torres que seus recursos permitem em posições aleatórias do mapa. Como se pode ver na Tabela 1 e no Gráfico 1 abaixo, ela normalmente perde nas primeiras ondas de inimigos e dificilmente atinge a sexta onda.

Tentativa	Onda
1	6
2	4
3	7
4	5
5	6
6	5
7	7
8	5
9	5
10	5
11	6
12	4
13	6
14	6
15	5

Tabela 1 - Número da onda alcançada a cada tentativa da distribuição aleatória. Fonte: PCS2050-11-CD-2015

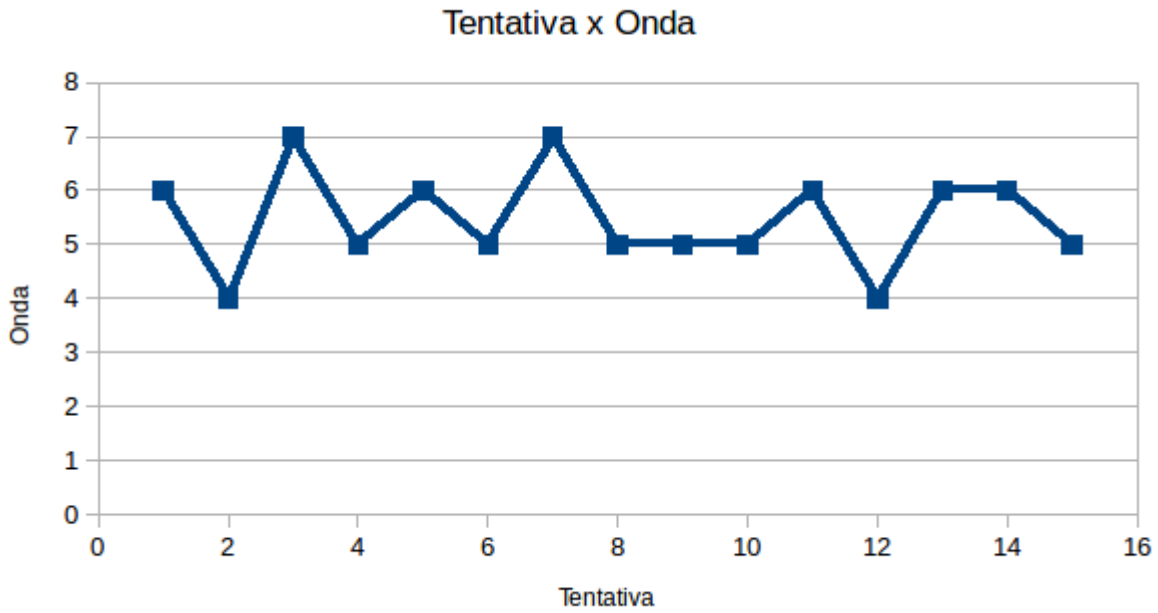


Gráfico 1 - Tentativa x Onda de distribuição aleatória.

6.2. Algoritmo de Pesos

O algoritmo de pesos mostra um desempenho bastante superior ao da IA de controle. No entanto, sua evolução não é constante. Como destina uma parte considerável de seus recursos a torres exploratórias colocadas em pontos aleatórios do mapa, o algoritmo pode ficar sem recursos para construir torres de acordo com o *ranking* por ele montado. Isso explica suas quedas de rendimento mesmo após ter conseguido alcançar desempenhos muito bons em tentativas anteriores.

Tentativa	Onda
1	4
2	5
3	9
4	14
5	11
6	10

7	11
8	11
9	10
10	18
11	13
12	9
13	13
14	28
15	18

Tabela 2 - Número da onda alcançada a cada tentativa do algoritmo de pesos. Fonte: PCS2050-11-CD-2015

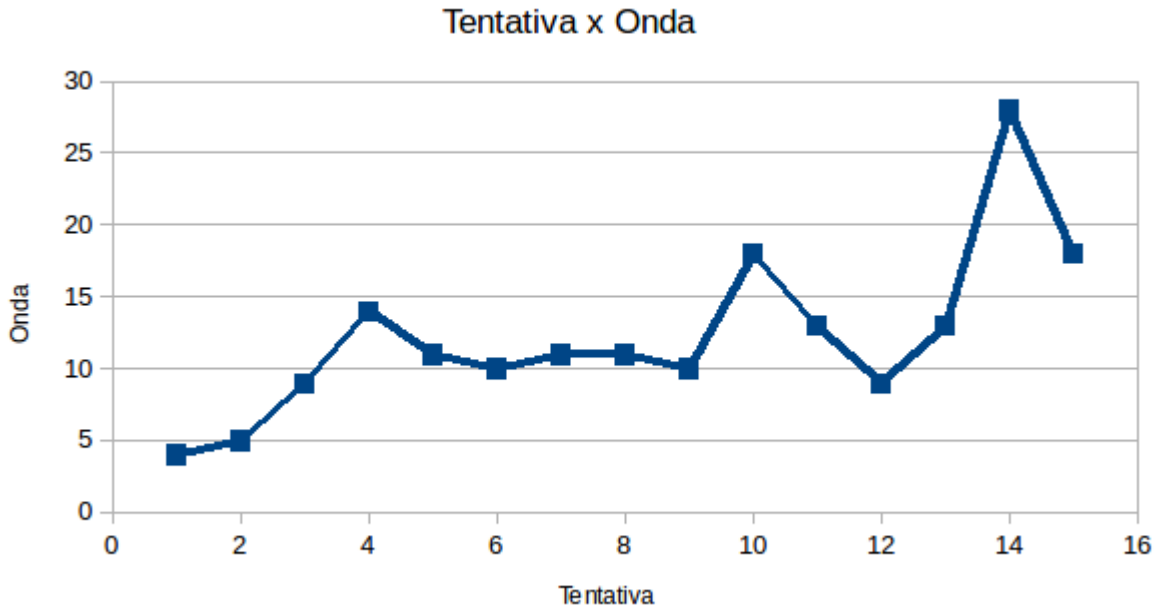


Gráfico 2 - Tentativa x Onda do algoritmo de pesos.

6.3. Algoritmo de Ordenação

O algoritmo de ordenação exibiu uma evolução clara e constante. Como ele repete a construção das torres úteis que construiu em tentativas anteriores, ele basicamente não piora os resultados já alcançados. Na verdade, ele tenta melhorá-los construindo mais cedo torres que tenham se destacado em turnos recentes de tentativas anteriores. O progresso quase linear desse algoritmo pode ser explicado pelo fato dele só explorar o mapa quando não conhece mais nenhuma posição útil onde construir novas torres.

Tentativa	Onda
1	5
2	5
3	7
4	8
5	8
6	9
7	10
8	11
9	12
10	14
11	13
12	17
13	18
14	20
15	22

Tabela 3 - Número da onda alcançada a cada tentativa do algoritmo de ordenação. Fonte: PCS2050-11-CD-2015

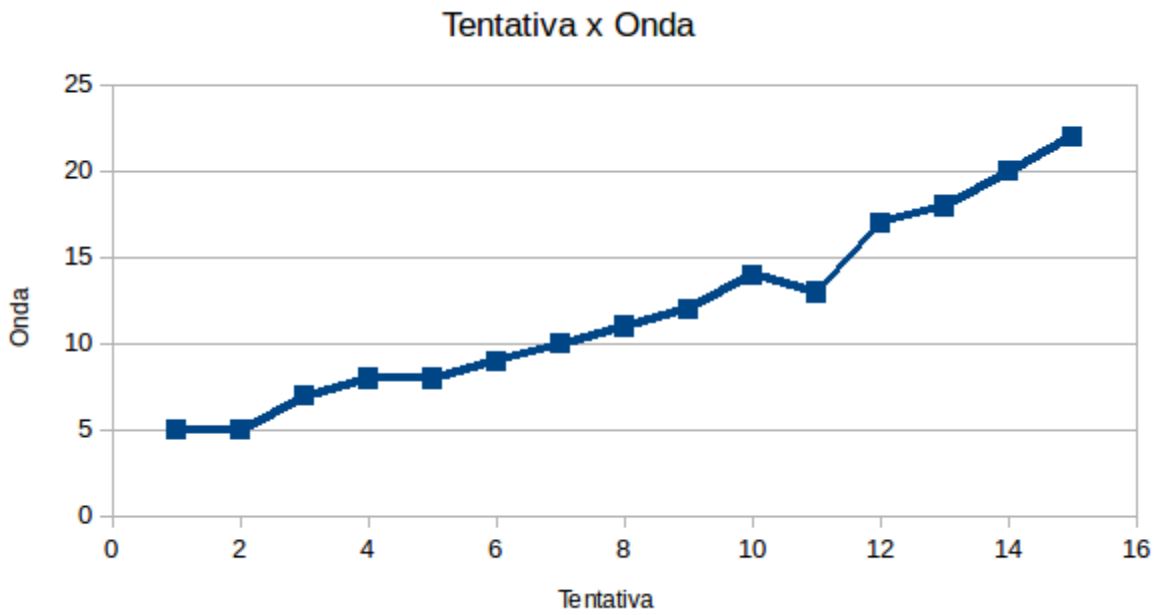


Gráfico 3 - Tentativa x Onda do algoritmo de ordenação.

6.4. Comparações entre os algoritmos

Como é fácil notar no gráfico abaixo, tanto o algoritmo de pesos quanto o de ordenação apresentaram uma tendência de evolução e foram capazes de “aprender” a jogar. Enquanto o algoritmo de distribuição de pesos é afetado pela sua aleatoriedade inerente e apresenta uma grande variação de resultados, o algoritmo de ordenação apresenta uma evolução quase linear.

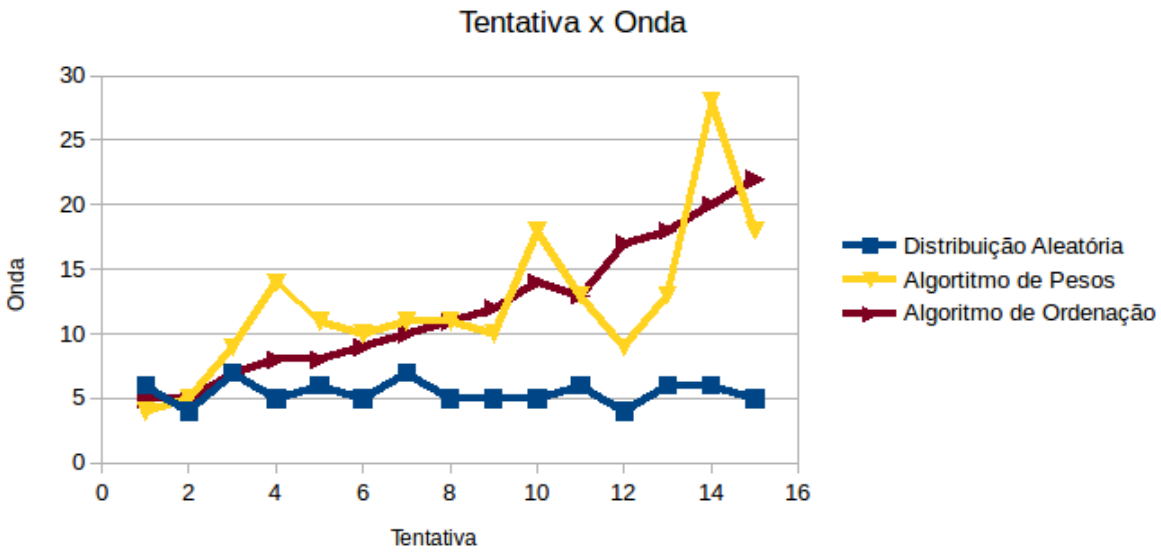


Gráfico 4 - Tentativa x Onda dos algoritmos de distribuição aleatória, pesos e ordenação.

6.5. Conclusões do capítulo de Resultados

O jogo base se mostrou uma plataforma sólida e customizável o suficiente para que todos os teste fossem realizados. A facilidade de se alterar o modo de funcionamento do jogo, como todo o mecanismo de envio de inimigos e sucessão de ondas, além dos mais diversos parâmetros de funcionamento do próprio jogo, foram de extrema utilidade no processo de tentativa e erro de balancear o jogo de modo que ele apresentasse um desafio adequado para os algoritmos implementados.

Já os algoritmos apresentados, como mostra a seção anterior, atingiram o objetivo inicial do trabalho: eles se adaptaram e aprenderam a jogar uma versão simples de um Tower Defense. Isso sem obter nenhuma informação prévia sobre o mapa ou sobre o funcionamento do jogo em si.

7. CONCLUSÃO

7.1. Considerações Finais

O objetivo desse trabalho foi construir uma inteligência artificial capaz de aprender a jogar um jogo simples do gênero *Tower Defense*, além de criar o próprio jogo base onde ela teria seu desempenho testado.

A primeira etapa foi criar o jogo propriamente dito. Tentamos construir uma arquitetura simples, mas facilmente modificável e expansível para que nos atendesse caso os requisitos do jogo viessem a mudar. Ao longo do desenvolvimento, foi observado seu potencial para servir como uma plataforma de estudo de jogos na área. Por isso, o grupo também adicionou aos seus objetivos deixa-lo mais acessível para os próximos que vierem a usa-lo.

Em paralelo, foram criados algoritmos e estratégias capazes de melhorar seu desempenho no jogo à medida que o jogassem. Várias das soluções propostas não foram bem sucedidas. Entretanto, foi possível observar, das propostas sucedidas, a evolução da inteligência desenvolvida, que, a cada tentativa, se mostrava mais eficiente.

7.2. Contribuição

Ao se pesquisar sobre Inteligência Artificial aplicada a jogos digitais é incomum encontrar trabalhos para o gênero de jogo de Tower Defense. Entre os trabalhos encontrados com fundamento em jogos, é predominante a escolha de jogos de estratégia de outros estilos para estudo e implementação de técnicas de IA. Por causa disso e da crescente popularidade que estes tipos de jogos alcançaram nos últimos tempos, o grupo se comprometeu a desenvolver e comparar diferentes algoritmos de Inteligência Artificial aplicados a jogos Tower Defense. O grupo também se comprometeu em construir o jogo base que seria o alicerce para as inteligências artificiais elaboradas.

Baseando-se nos resultados e conclusões discutidos em seções prévias, fica claro que é possível implementar algoritmos de Inteligência Artificial que se adaptem

ao estilo de jogo de Tower Defense. Com isso, o grupo pôde comprovar que há espaço para utilização de IA neste gênero de jogo.

O grupo também cede a utilização do jogo base por qualquer pessoa que queira codificar novos algoritmos de IA ou estender o próprio jogo. O jogo desenvolvido é altamente customizável e possui módulos principais que facilitam a criação de novos tipos de inimigos, modificação das torres existentes ou outras customizações. Além disso, o jogo já disponibiliza recursos para medir a eficiência da inteligência desenvolvida sem maiores alterações.

7.3. Trabalhos Futuros

Os algoritmos de Inteligência Artificial testados neste trabalho são apenas um conjunto pequeno frente a todos algoritmos criados para jogos de estratégia. Sendo assim, o grupo convida a quem tiver interesse em expandir essa área, que aplique outros modelos de IA sobre jogos de Tower Defense.

REFERÊNCIAS

AVERY, P. et al. **Computational intelligence and tower defence games**. In: IEEE Congress on Evolutionary Computation'11. Nevada: [s.n.], 2011. p. 1084–1091.

CAELLUM. **Java e Orientação a Objetos:** Apostila do curso FJ-11. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/>. Acesso em: 20 nov. 2015.

CASS, Stephen. **The 2015 Top Ten Programming Languages**: New languages enter the scene, and big data makes its mark. 2015. Disponível em: <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>. Acesso em: 20 nov. 2015.

CHACON, Scott; STRAUB, Ben. **Pro Git: Everything you need to know about Git.** 2. ed.: Apress, 2014.

LIBGDX. **Introduction**. 2013. Disponível em: <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>. Acesso em: 20 nov. 2015.

MCCARTHY, J. **Basic Questions.** 2012. Disponível em: <http://www-formal.stanford.edu/jmc/whatisai/node1.html>. Acesso em: 22 de novembro de 2015.

ROHTKRANTZ, L.J.M.; KOPPELAAR, H.; SPRONCK, P.H.M.. **Improving Adaptive Game AI With Evolutionary Learning**. 2004. 45 f. Tese de Mestrado, Faculty Of Media & Knowledge Engineering Delft University Of Technology, Delft, 2004.

RUMMEL, Paul A.. **Adaptive AI to play tower defense game**. In: COMPUTER GAMES (CGAMES), 16., 2011, Louisville. Teste. Louisville: CGames, 2011. p. 38 - 40.

VIEIRA, D. **Scrum: A Metodologia Ágil Explicada de uma forma Definitiva.** 2004 Disponivel em: <http://www.mindmaster.com.br/scrum/>. Acesso em: 22 de nov. 2015.

LEITE, F. LOPES. T, LAMPERT, H. **PCS2050-11-CD-2015.** São Paulo. 2015.

REFERÊNCIAS DE IMAGENS

KINGDOM RUSH. **Screenshot do jogo.** Disponível em: <http://www.kingdomrush.com/home.html>. Acesso em: 22 nov. 2015.

TRELLO. **Backlog e andamento das atividades.** Disponível em: <http://www.trello.com> Acesso em: 27 nov. 2015.

GLOSSÁRIO

IDE: *Integrated Development Environment,* programa que reúne ferramentas que apoiam e agilizam o processo de desenvolvimento de software.

Motor de Jogo: Programa de computador e/ou conjunto de bibliotecas, para simplificar e abstrair o desenvolvimento de jogos eletrônicos.

NPC: Acrônimo para Non-Player Character. Personagens que o jogador não pode controlar durante o jogo.

Framework: Abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

APÊNDICE

PCS2050-11-CD-2015

O CD possui 4 arquivos, esta monografia (PCS2050-11-Monografia-2015.pdf), os logs referentes aos testes realizados com o algoritmo de distribuição aleatória (logs/aleatoria.txt), com o algoritmo de pesos (logs/pesos.txt), com o algoritmo de ordenação (logs/ordenacao.txt).