



UNIVERSIDADE ESTADUAL DE MARINGÁ

DEPARTAMENTO DE INFORMÁTICA

Bacharelado em Ciência da Computação



Estratégias Algorítmicas em Jogos de Cartas Competitivos

Maringá, 2024

Universidade Estadual de Maringá – Departamento de Informática

Av. Colombo, 5790 – Bloco C56, Zona 7 – CEP: 87020-900

Fone: (44) 3011-4324/3011-4219

Maringá – PR



Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática



Estratégias Algorítmicas em Jogos de Cartas

Competitivos

VICTOR HUGO DO NASCIMENTO BUENO

TCC-2024

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Maringá, como parte dos requisitos necessários à obtenção do título de Bacharel em NOME DO CURSO.

Orientador: WAGNER IGARASHI

Banca: IGOR DA PENHA NATAL

Banca: ANDRE FELIPE RIBEIRO CORDEIRO

Agradecimentos

Agradeço profundamente ao meu orientador, Prof. Dr. Wagner Igarashi, pelo apoio essencial e orientação ao longo desta jornada. Agradeço também aos meus amigos que estiveram me apoiando incondicionalmente durante o desenvolvimento deste projeto. Por fim, sou grato à minha família, especialmente ao meu avô, cujo apoio, sabedoria e carinho foram fundamentais para minha trajetória e para a realização deste trabalho.

Resumo

Nos últimos anos, o mundo dos jogos eletrônicos tem testemunhado um notável aumento em sua popularidade e influência cultural. Com o avanço da tecnologia e a disseminação de dispositivos eletrônicos, os jogos tornaram-se parte do cotidiano das pessoas. Dentre os diversos gêneros de jogos, os jogos competitivos se destacam por proporcionar aos jogadores a oportunidade de medir suas habilidades contra adversários. Nesses jogos, os jogadores buscam constantemente por novos desafios, e a qualidade dos adversários desempenha um papel crucial na experiência do jogador. Nos primórdios dos jogos eletrônicos, o comportamento dos adversários controlados pelo computador eram programados sem “inteligência”, baseando-se em padrões determinísticos, o que limita a capacidade de adaptação do computador a novas situações de jogo e em oferecer desafios dinâmicos para os jogadores. Com os avanços na Inteligência Artificial (IA), especialmente em técnicas de aprendizagem de máquina, tornou-se possível desenvolver adversários mais adaptáveis e inteligentes. A aprendizagem por reforço, uma técnica de IA, permite que os adversários aprendam e melhorem com base em interações. Assim, o objetivo deste trabalho é explorarmos a aplicação de técnicas de IA para programar o comportamento do computador adversário no jogo competitivo desenvolvido. E assim realizarmos uma análise do desempenho da abordagem implementada, buscando aprimorar a jogabilidade e também contribuir com o avanço do conhecimento de IA em jogos do mesmo gênero.

Palavras-chave: Desenvolvimento de Jogo; Cartas; Inteligência Artificial; Gamedev.

Abstract

In recent years, the world of video games has witnessed a remarkable increase in its popularity and cultural influence. With the advancement of technology and the spread of electronic devices, games have become part of people's daily lives. Among the various game genres, competitive games stand out for providing players the opportunity to test their skills against opponents. In these games, players are constantly seeking new challenges, and the quality of the opponents plays a crucial role in the player's experience. In the early days of video games, the behavior of computer-controlled opponents was programmed without "intelligence," relying on deterministic patterns, which limited the computer's ability to adapt to new game situations and offer dynamic challenges to players. With advancements in Artificial Intelligence (AI), particularly in machine learning techniques, it has become possible to develop more adaptable and intelligent opponents. Reinforcement learning, an AI technique, allows opponents to learn and improve based on interactions. Thus, the aim of this work is to explore the application of AI techniques to program the behavior of the computer opponent in the developed competitive game, and to analyze the performance of the implemented approach, aiming to enhance gameplay and contribute to the advancement of AI knowledge in games of the same genre.

Keywords: Game Development; Cards; Artificial Intelligence; Gamedev.

Sumário

1. Introdução.....	10
2. Fundamentação Teórica.....	12
2.1. Inteligência Artificial.....	12
2.2. Evolução da Inteligência Artificial em Jogos.....	12
2.3. Métodos de IA.....	15
2.3.1. Máquinas de Estados Finitos.....	15
2.3.1. Árvores de Comportamento.....	16
2.3.2. Busca de Monte Carlo em Árvores.....	17
2.3.3. Aprendizagem por Reforço.....	19
3. Especificação do jogo.....	20
3.1. Conceito do Jogo.....	20
3.2. Regras do Jogo.....	21
3.3. Cartas.....	23
3.3.1. Cartas de Poder.....	23
3.3.2. Cartas de Efeito.....	26
4. Implementação.....	27
4.1. Ferramentas Utilizadas.....	27
4.2. Estrutura do Projeto.....	29
4.3. Lógica do Jogo e Suas Classes.....	30
4.3.1. Card.....	30
4.3.2. PlayerEntity.....	32
4.3.3. Battle.....	33
4.4. Sprites.....	34
4.5. Telas.....	38
4.6. Áudio.....	45
4.7. Interação Tela-Jogo.....	46
4.8. Abordagens de IA.....	47
4.8.1. IA Focada na Maximização de Poder.....	49

4.8.2. IA Focada na Conservação de Mana.....	50
5. Metodologia e Configurações.....	51
6. Análise dos Resultados.....	52
6.1. Resultados Médios.....	52
7. Dificuldades e Desafios.....	54
8. Conclusão.....	55
Referências Bibliográficas.....	56
Apêndices.....	58
Apêndice A - Implementações.....	58
A.1 - Classe Estatística de Batalha.....	58
A.2 - Simulação de Batalha entre IAs.....	58
A.3 - Service de Estatísticas de Batalhas.....	60

Lista de Figuras

Figura 2.2.1: Imagem do jogo Pong.....	12
Figura 2.2.2: Imagem do jogo Space Invaders.....	12
Figura 2.2.3: Imagem do jogo Zork.....	13
Figura 2.3.1: Exemplo de um FSM.....	14
Figura 2.3.2: Exemplo de uma BT.....	16
Figura 3.1: Imagem do jogo Pokémon TCG.....	19
Figura 3.2.1: Diagrama de fluxo do jogo.....	20
Figura 4.1.1: Ambiente de Edição de Imagens Photopea.....	26
Figura 4.1.2: IDE IntelliJ IDEA.....	26
Figura 4.7.1: Fluxograma Interação Tela-Jogo.....	45

Lista de Quadros

Quadro 3.3.1.1: Vantagens e desvantagens.....	23
Quadro 3.3.1.2: Exemplo Interação ÁGUA-FOGO em Cartas de Poder.....	24
Quadro 3.3.1.2: Exemplo de Cartas de Batalha em campo aplicando vantagens/desvantagens...	25
Quadro 4.2.1: Estrutura do Projeto.....	28
Quadro 4.3.1.1: Trecho de Código simplificado para Classe Card.....	29
Quadro 4.3.1.2: Trecho de Código simplificado para Classe CardManager.....	30
Quadro 4.3.2.1: Trecho de Código simplificado para Classe PlayerEntity.....	31
Quadro 4.3.3.1: Trecho de Código simplificado para Classe Battle.....	33
Quadro 4.4.1: Sprites de Tipo de Cartas.....	34
Quadro 4.4.2: Sprites de Tipo de Elementos.....	34
Quadro 4.4.3: Sprites de Status.....	35
Quadro 4.4.4: Sprites de Personagens.....	36
Quadro 4.5.1: Tela Inicial.....	38
Quadro 4.5.2: Tela de Batalha.....	39
Quadro 4.5.3: Painel de Turno.....	39
Quadro 4.5.4: Painel do Jogador.....	40
Quadro 4.5.5: Caixa de Texto de Batalha.....	41
Quadro 4.5.6: Caixa de Texto de Batalha no modo de Seleção de Ação.....	42
Quadro 4.5.7: Caixa de Texto de Batalha no modo de Seleção de Carta.....	43
Quadro 4.6.1: jsfxr - 8 bit sound maker and online sfx generator.....	44
Quadro 4.8.1.1: Pseudocódigo primeira abordagem IA.....	47
Quadro 4.8.2.1: Pseudocódigo segunda abordagem IA.....	48
Quadro 5.1: Cartas utilizadas no deck de testes.....	49

1. Introdução

A indústria de jogos eletrônicos tem experimentado um crescimento exponencial nas últimas décadas, tornando-se uma das principais formas de entretenimento mundial. Segundo o relatório da Newzoo de 2023, o número de jogadores foi estimado em cerca de 3,31 bilhões, com os jogadores de dispositivos móveis sendo a maior influência nesse aumento. Esse fenômeno reflete não apenas o avanço tecnológico, mas também a crescente popularização dos jogos eletrônicos na cultura cotidiana.

O sucesso de um jogo está diretamente ligado à experiência do jogador, um conceito amplamente discutido na área de design de jogos. Segundo Schell (2019), a experiência do jogador emerge da interação entre os elementos que fazem parte do jogo, como: interface, mecânicas e jogabilidade. A Interface refere-se aos controles e à apresentação visual do jogo. As regras ou mecânicas, definem como os jogadores interagem com o ambiente, estabelecendo desafios e recompensas que mantêm o jogador engajado (Koster, 2013). Por fim, a jogabilidade abrange a resposta do jogo às ações do jogador, sendo um fator importante para a sensação de imersão e diversão (Juul, 2005).

A capacidade do computador de adaptar-se de maneira inteligente não era vista nos primeiros jogos eletrônicos, onde os adversários possuíam comportamentos determinísticos, o que algumas vezes comprometia a jogabilidade, tornando os desafios mais simples e previsíveis.

A Inteligência Artificial é um ramo da ciência que tem se relacionado com os jogos ao longo dos anos, permitindo a melhora no quesito da experiência do jogador. Nesse sentido, a IA tem usado os jogos como um campo para aplicação de técnicas, visto que são ambientes formais, altamente restritos e complexos para a tomada de decisões (Yannakakis, 2018).

Historicamente, as primeiras pesquisas em IA para jogos foram realizadas em jogos de tabuleiro clássicos, como Damas e Xadrez, onde o objetivo era o de desenvolver agentes capazes de vencer os melhores jogadores humanos. Ao longo dos anos, diversas IAs obtiveram um sucesso notável, como Chinook, que em 1994 derrotou o campeão mundial de damas Marion Tinsley (Schaeffer et al., 2007). Outro exemplo é o Deep Blue, da IBM, que venceu o mestre de xadrez Garry Kasparov em 1997 (Campbell; Hoane Jr.; HSU, 2002). Mais recentemente, o AlphaGo, desenvolvido pela DeepMind, superou o campeão mundial de Go, Ke Jie, demonstrando a capacidade da IA em dominar jogos estratégicos (Silver et al., 2016).

Além dos jogos de tabuleiro clássicos, a IA também tem sido amplamente utilizada em outras áreas dos jogos eletrônicos, como na geração de conteúdo. Um exemplo disso é o jogo “Rogue” (Toy and Wichmann, 1980), onde as *dungeons* e a posição de criaturas e itens são geradas algoritmicamente toda vez que um novo jogo começa. Outro exemplo de aplicação da IA em jogos é o trabalho desenvolvido pela Google DeepMind em 2014, onde algoritmos foram projetados para aprender como jogar diversos jogos clássicos do ATARI 2600, imitando comportamentos humanos (Yannakakis, 2018).

Dessa forma, a contínua evolução da Inteligência Artificial tem ampliado a possibilidade no design de jogos, desde a criação de adversários desafiadores até a geração procedural de conteúdo. Com o avanço das técnicas de IA, os jogos têm se tornado mais dinâmicos e adaptativos, oferecendo experiências cada vez mais realísticas. Portanto, neste trabalho, exploraremos o uso de diferentes abordagens para uma IA, do jogo de cartas competitivo desenvolvido. Dessa forma, o objetivo deste trabalho é demonstrar como essas técnicas podem ser aplicadas para desenvolver adversários inteligentes e melhorar a imersão do jogador com o jogo.

2. Fundamentação Teórica

2.1. Inteligência Artificial

A Inteligência Artificial (IA) é um campo universal, onde procuramos sistematizar e automatizar tarefas intelectuais. Apesar da definição apresentada, é difícil delimitar precisamente o termo IA, pois diferentes autores a descrevem sob perspectivas diferentes (Russel; Norvig, 2022).

Por exemplo, alguns pesquisadores definem IA como a capacidade de uma máquina em imitar a inteligência humana. Outros a consideram como a ciência e engenharia de criar máquinas inteligentes, especialmente programas de computador capazes de realizar tarefas que normalmente requerem a inteligência humana. Essa diversidade de definições reflete a multidisciplinaridade do campo, abrangendo áreas como ciência da computação, filosofia, neurociência, psicologia, entre outras (Russel; Norvig, 2022).

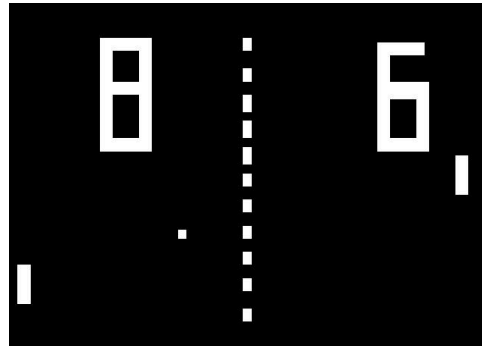
Embora essa dificuldade conceitual exista, é evidente que esse ramo da ciência tem aplicações em diversas áreas, como medicina, reconhecimento de imagens, traduções, reconhecimento de fala e, em especial, na área de jogos.

2.2. Evolução da Inteligência Artificial em Jogos

Conforme explorado na introdução, a relação entre IA e a área de jogos tem evoluído significativamente ao longo dos anos. Nesta seção, vamos aprofundar essa relação “Jogos-IA” e como ela se desenvolveu ao longo dos anos.

Pong foi um dos jogos pioneiros de fliperama e introduziu um tipo de inteligência artificial “primitiva” no universo dos jogos eletrônicos. O jogo consistia em um simples jogo de tênis de mesa, onde o jogador controlava uma raquete verticalmente e tinha como objetivo refletir a bola de modo a fazê-la atravessar o campo do adversário.

Figura 2.2.1: Imagem do jogo Pong.

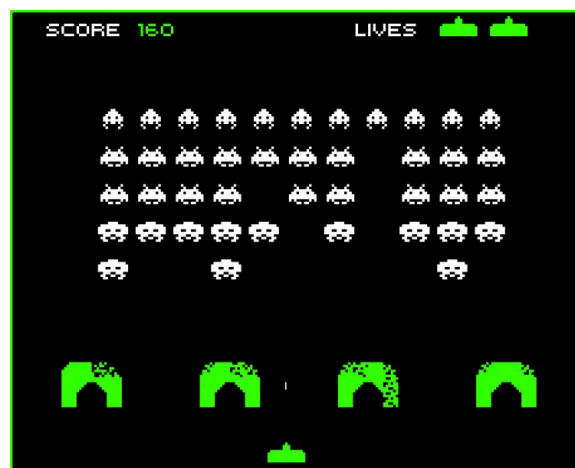


Fonte: PONG (Atari, 1972) - Bojogá

A IA do Pong executava movimentos verticais na tentativa de acertar a bola com base em padrões predefinidos. Dessa forma, não havia aprendizado envolvido por parte da IA, tornando o movimento previsível e repetitivo (Assaf, 2023).

Segundo Assaf, outro jogo que foi essencial no desenvolvimento inicial da IA em jogos foi o Space Invaders. O jogo apresentava uma fila de alienígenas que se moviam de um lado para o outro e desciam lentamente em direção ao jogador, cujo objetivo era o de eliminar os alienígenas com sua nave que atirava lasers.

Figura 2.2.2: Imagem do jogo Space Invaders.



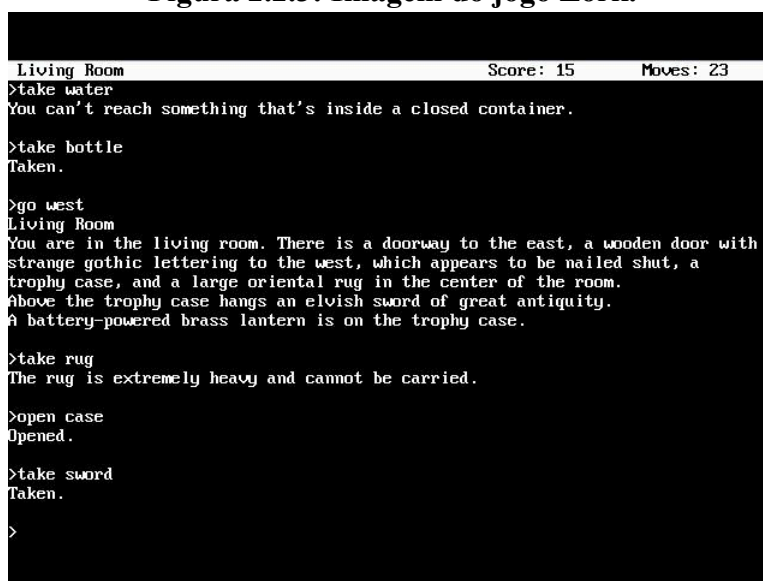
*Fonte: The Original 'Space Invaders' Is a Meditation on 1970s America's Deepest Fears
Smithsonian*

A IA do Space Invaders, assim como a do Pong, seguia um conjunto de regras predefinidas. A principal diferença entre os dois jogos, se dá no caso do Space Invaders com a

introdução de elementos mais desafiadores como “*speed up*” e aleatoriedade no movimento dos alienígenas (Assaf, 2023).

Com o avanço da tecnologia em meados de 1980, os jogos começaram a se mover em direção a interações mais complexas, como visto em Zork, um jogo baseado em texto que empregava algoritmos de processamento de linguagem natural (PLN) para interpretar os comandos dos jogadores (Ex: “go to north”). Embora os algoritmos de Zork fossem simples para os padrões atuais, eles foram revolucionários para a época, abrindo caminho para interações mais sofisticadas do jogador com o ambiente do jogo (Assaf, 2023).

Figura 2.2.3: Imagem do jogo Zork.



Fonte: Zork I: The Great Underground Empire - Gameplay

Outro marco importante na evolução da IA em jogos foi o lançamento de Spore em 2008. Ao contrário de jogos tradicionais, que dependiam de designers para criar manualmente o conteúdo, Spore empregava uma abordagem baseada em IA para gerar criaturas e mundos de forma procedural, tornando cada mundo uma experiência única para o jogador (Hillary, 2023).

De acordo com Hillary (2023), os anos de 2010 marcaram um ponto importante para as IAs em jogos, com uma mudança de paradigma decorrente da introdução do aprendizado de máquina. Nesse período, os desenvolvedores começaram a incorporar algoritmos capazes de aprender e evoluir com base nas interações dos jogadores, resultando em experiências de jogo mais imersivas e desafiadoras.

A evolução da Inteligência Artificial em jogos eletrônicos demonstra um avanço significativo desde os primeiros jogos com comportamentos determinísticos até a era moderna, onde técnicas mais sofisticadas como aprendizado de máquina, processamento de linguagem natural e entre outras técnicas são utilizadas.

Esses avanços refletem a capacidade da IA de não apenas replicar a natureza do comportamento humano, mas também de aprimorar e personalizar a experiência do jogador de maneiras que antes não eram possíveis, tornando os jogos capazes de oferecer desafios mais complexos e realistas.

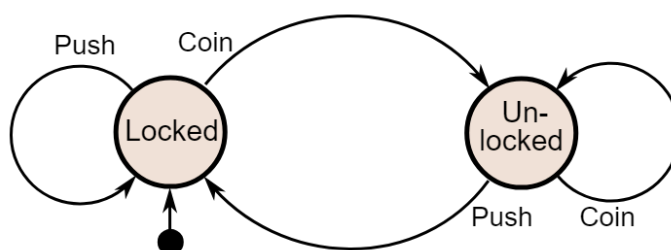
2.3. Métodos de IA

Conforme visto na seção anterior, à medida que os jogos evoluíram, métodos de IA mais sofisticados foram utilizados no contexto de jogos. Nesta seção, exploraremos os principais métodos que têm sido implementados no desenvolvimento de jogos eletrônicos, desde os mais tradicionais até os mais avançados.

2.3.1. Máquinas de Estados Finitos

A Máquina de Estados Finitos, ou também do inglês *Finite State Machine* (FSM), foi o método de IA predominante para o controle e tomada de decisões de personagens não jogáveis até meados de 2000 (Yannakakis, 2018). O FSM é representado por um grafo, onde os nós representam estados e as conexões entre os nós são chamados de transições. A máquina de estados pode estar apenas em um estado por vez, e o estado atual pode mudar para outro se a condição associada à transição for satisfeita.

Figura 2.3.1: Exemplo de um FSM.



Fonte: 9.1.1: [*Finite-State Machine Overview - Engineering LibreTexts*](#)

Cada estado no FSM encapsula um comportamento ou ação específica de um personagem não jogável (do inglês, non-player character - NPC), como “perseguir o jogador”, “fugir” ou “patrulhar”. As transições entre os estados são ativadas por condições particulares, por exemplo, “quando o jogador estiver próximo”, permitindo que os NPCs realizem ações variadas de forma lógica e previsível, baseando suas decisões em regras pré-definidas.

As Máquinas de Estados Finitos são relativamente simples de serem modeladas e implementadas, o que as torna uma escolha popular para o desenvolvimento de IA em jogos. Entretanto, essa simplicidade vem com a falta de flexibilidade e adaptabilidade. Como os comportamentos dos NPCs são definidos por estados e transições pré-determinadas, eles tendem a se tornar previsíveis ao longo do tempo.

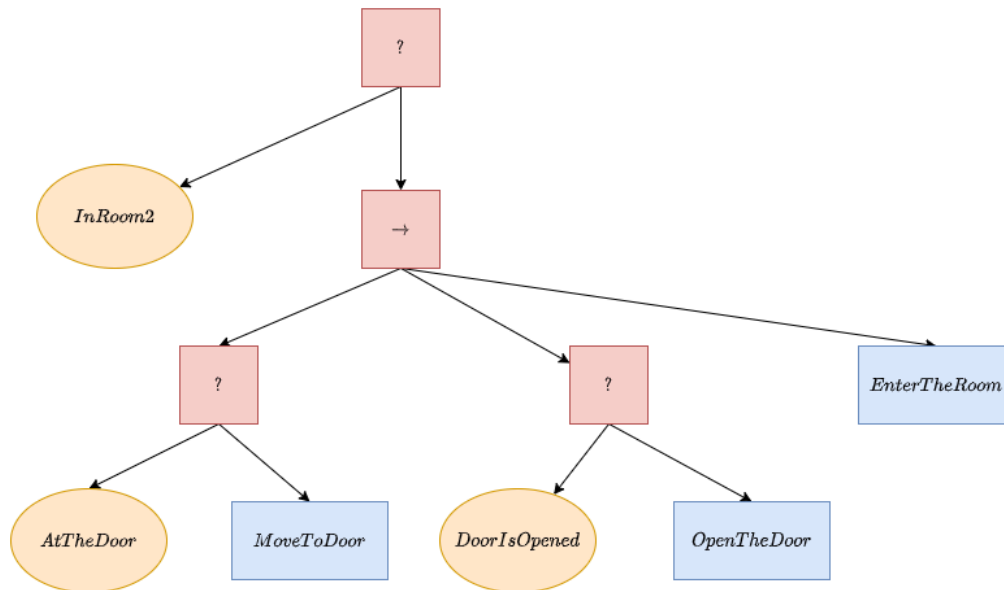
2.3.1. Árvores de Comportamento

As árvores de comportamento, ou *Behaviour Trees* (BTs) em inglês, surgem como uma alternativa modular e mais flexível para modelar o comportamento de agentes em jogos. Diferentemente das Máquinas de Estados Finitos (FSMs), as quais utilizam estados para representar o comportamento dos NPCs, as BTs são compostas por nós que representam tarefas ou comportamentos (Yannakakis, 2018).

As BTs são organizadas de forma hierárquica, com uma estrutura em árvore composta por um nó raiz, nós pais e nós filhos. A execução começa pelo nó raiz, o qual inicia o processo de tomada de decisão, e segue através dos nós filhos conforme a lógica definida na árvore.

Uma das principais vantagens das BTs é sua modularidade, permitindo a criação de comportamentos complexos a partir da combinação de tarefas simples. Um exemplo de BTs pode ser observado na figura 2.3.2. Além disso, por apresentarem facilidade no design, teste e depuração, as BTs tornaram-se uma escolha atraente no desenvolvimento de jogos. Segundo Yannakakis um dos casos de sucesso foi a aplicação das BTs em jogos como Halo 2 (Microsoft Game Studios, 2004) e Bioshock (2K Games, 2007).

Figura 2.3.2: Exemplo de uma BT.



Fonte: *Designing AI Agents' Behaviors with Behavior Trees* | by Debby Nirwan | *Towards Data Science*

A importância das Árvores de Comportamento na modelagem de IA para jogos é tamanha que muitas das principais *engines* de desenvolvimento de jogos, como a Unreal Engine, incorporam suporte nativos para BTs (Unreal Engine, 2024).

2.3.2. Busca de Monte Carlo em Árvores

A maior parte dos problemas de IA são relacionados a problemas de busca, os quais podem ser solucionados encontrando o “melhor” plano, caminho, modelo, função, etc. Dentre os algoritmos de busca em árvores, o Algoritmo de Monte Carlo (*Monte Carlo Tree Search* - MCTS) se destaca, particularmente em ambientes onde a busca exaustiva é impraticável devido ao vasto espaço de estados possíveis. Um exemplo notável é o jogo Go, cujo tabuleiro de 19x19 cria um número de configurações tão vasto que torna impraticável o uso de algoritmos de busca tradicionais, como o Minimax (Russel; Norvig, 2022).

O MCTS é estruturado em quatro etapas principais: Seleção, Expansão, Simulação e Retropropagação. Na etapa de seleção, o algoritmo percorre a árvore de decisão a partir da raiz, escolhendo um movimento com base em uma política de seleção que equilibra a exploração de novos caminhos e a exploração de jogadas promissoras. Esse processo continua

até que ele alcance um nó que ainda não foi explorado, momento em que a próxima etapa, a Expansão, é iniciada.

Como política de seleção, geralmente é utilizado o critério UCB (*Upper Confidence Bound*). Esse critério calcula um valor para cada nó da árvore que foi explorado, levando em consideração tanto a média de recompensas obtidas até o momento quanto a incerteza associada a essas recompensas. De forma geral, ele assegura que o algoritmo explore novas áreas da árvore de decisão quando apropriado, ao mesmo tempo em que se concentra em áreas que demonstram potencial.

Seja um nó n na árvore, $UCB(n)$ é dado por:

$$UCB(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(\text{Parent}(n))}{N(n)}}$$

Onde:

- $\frac{U(n)}{N(n)}$ é a média de recompensas obtidas pelo nó n .
- $N(n)$ é o número de vezes que o nó n foi visitado.
- C é uma constante que ajusta o grau de exploração do algoritmo (quanto maior o valor de C mais o algoritmo tende a explorar novos nós).
- $N(\text{Parent}(n))$ é o número de vezes que o nó pai de n foi visitado

Após a seleção, o MCTS prossegue para a etapa de Expansão. Nesta fase, o algoritmo adiciona um novo nó à árvore de decisão, correspondente a uma ação ainda não explorada. Uma vez que um novo nó é adicionado, o algoritmo passa para a etapa de Simulação.

Na etapa de Simulação, o algoritmo escolhe ações a partir do novo nó filho até alcançar um nó folha. Essas escolhas são realizadas de acordo com uma política que pode variar conforme o problema e a implementação. Em muitos casos, a política adotada durante a simulação é a de escolher as ações de forma aleatória, permitindo ao algoritmo explorar uma ampla variedade de possíveis resultados.

Com o fim da simulação ao chegar em um nó folha, um resultado final é obtido, que reflete o desfecho do caminho seguido durante a simulação. Esse resultado é, então, utilizado na etapa de Retropropagação, onde ele é transmitido de volta através dos nós que foram

visitados ao longo da simulação. Durante esse processo, cada nó visitado tem suas estatísticas atualizadas, como por exemplo a média de recompensas e número de visitas.

Essas quatro etapas são repetidas até que um número de iterações seja alcançado ou um limite de tempo seja atingido. No final, o algoritmo seleciona a ação que retorna o maior valor de *UCB*, apontado para a estratégia mais promissora identificada.

Um exemplo interativo da aplicação do MCTS pode ser encontrado no simulador de Tic-Tac-Toe desenvolvido por Vinícius Garcia. Esse simulador permite observar o funcionamento do MCTS em tempo real, mostrando a expansão da árvore de decisões e como o algoritmo explora diferentes jogadas. O usuário também pode ajustar alguns parâmetros como número de iterações para testar a eficiência do algoritmo com diferentes configurações (Garcia, 2020).

2.3.3. Aprendizagem por Reforço

A aprendizagem por Reforço (*Reinforcement Learning* - *RL*) é um paradigma de aprendizado baseado na tentativa e erro, no qual um agente interage com um ambiente e aprende a tomar decisões por meio de recompensas e penalidades. Diferente do aprendizado supervisionado, a RL não requer dados para treinamento, permitindo que o agente descubra estratégias conforme a experimentação (Sutton; Barto, 2018).

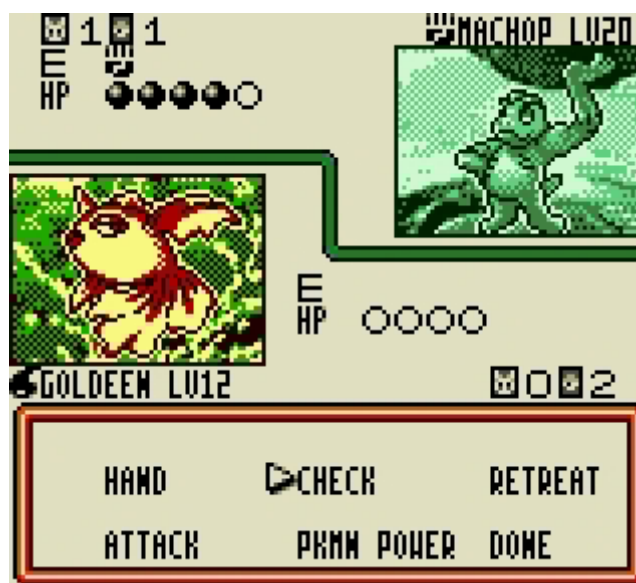
O **Monte Carlo Tree Search (MCTS)**, abordado na subseção anterior, pode ser utilizado como uma ferramenta dentro de sistemas de Aprendizagem por Reforço. O MCTS auxilia o agente a explorar e avaliar diferentes possíveis decisões durante o processo de aprendizagem. Um exemplo da utilização do MCTS é o **AlphaGo**, desenvolvido pela DeepMind, que utilizou MCTS em conjunto com RL e redes neurais para aprender políticas eficientes no jogo Go, como mencionado na seção de introdução deste trabalho.

Com esse embasamento teórico sobre técnicas utilizadas na criação de agentes inteligentes, é possível compreender a importância dessas ferramentas para a criação de adversários desafiadores. Apesar desses algoritmos terem sido discutidos ao longo do trabalho, devido a limitações de tempo, não foi possível implementá-los no desenvolvimento do jogo. No entanto, esses conceitos ainda servem como base para futuras melhorias e adições no jogo. Na seção seguinte, será apresentada a especificação do jogo, detalhando suas mecânicas, inspirações e objetivos.

3. Especificação do jogo

O desenvolvimento deste jogo de cartas foi inspirado em clássicos como *Pokémon Trading Card Game* (TCG) para Game Boy Color e no minigame *Card-Jitsu* do jogo *Club Penguin*. Ambos os jogos são baseados em turnos, onde as tomadas de decisões realizadas pelo jogador em cada rodada impactam significativamente no rumo da partida.

Figura 3.1: Imagem do jogo Pokémon TCG



Fonte: *Pokémon Trading Card Game (video game)* - Wikipédia

3.1. Conceito do Jogo

O jogo funciona em um sistema de turnos onde cada jogador possui seu próprio *deck* de cartas, muito similar ao estilo do **Magic: The Gathering** e **Yu-Gi-Oh!**, dois jogos de cartas populares. O objetivo do jogo é reduzir os pontos de vida do oponente a zero, utilizando estratégias que envolvem a combinação de cartas disponíveis na mão. A principal diferença do jogo em relação a esses títulos é o sistema de mana, que limita a utilização das cartas, e a dinâmica de combate que se distingue pela maneira como as cartas de poder são posicionadas.

Enquanto em jogos como Yu-Gi-Oh!, as cartas monstros (semelhante às cartas de poder no contexto do jogo) são colocadas em campo e seus efeitos podem ser vistos durante o turno, no jogo elaborado, as cartas de poder são posicionadas em campo, mas permanecem invisíveis até a fase de resolução de combate, após o término do turno de ambos os jogadores. Na fase de resolução de combate, as cartas de poder de ambos os jogadores são reveladas

simultaneamente, levando em consideração o poder de cada uma, o que determina o vencedor de cada batalha.

Além disso, cada *deck* contém dois tipos de cartas principais: **Cartas de Poder** e **Cartas de Efeito**. As cartas de poder são responsáveis por causar dano ao oponente, enquanto as cartas de efeito possuem habilidades predefinidas que podem alterar a dinâmica do jogo.

3.2. Regras do Jogo

Como configuração inicial, ambos os jogadores começam a partida com seis pontos de vida, zero de mana e com seis cartas em mãos, respeitando o limite máximo de seis cartas. Esses valores foram definidos com base em testes e considerações sobre balanceamento, dinâmica da partida e inspiração em outros jogos de cartas.

A escolha de seis pontos de vida foi determinada para garantir que as partidas tivessem uma duração moderada, sem serem excessivamente longas. Durante os testes, verificou-se que esse número mantinha as partidas dinâmicas, evitando que as partidas fossem monótonas e extensas.

O limite de seis cartas na mão foi inspirado em jogos que possuem mecânicas semelhantes de compra e gerenciamento de cartas, como o Card-Jitsu. Esse número permite que o jogador tenha opções estratégicas suficientes sem tornar o jogo dependente de sorte ou sobrecarregado com muitas possibilidades.

Em relação ao zero de mana no início do jogo, essa configuração foi escolhida com o objetivo de tornar o início da partida menos impactante. Com isso, os jogadores precisam construir suas estratégias gradualmente, sem a possibilidade de utilizar cartas e efeitos muito poderosos no início do jogo.

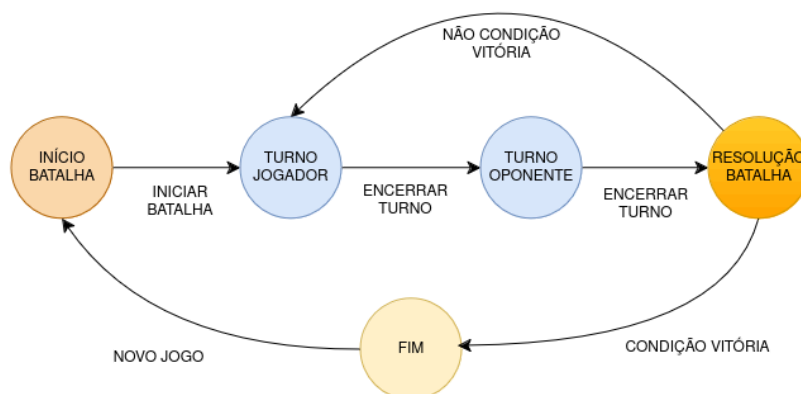
No início do turno de cada jogador, uma nova carta é comprada de seu respectivo *deck* e um ponto de mana é recuperado, tendo um limite máximo de três de mana. Durante o turno de cada jogador, suas ações são limitadas a utilizar as cartas em mãos ou encerrar seu turno. Cada carta possui um custo de mana, e desde que o jogador possua esse valor, ele pode realizar combinações de jogadas, como posicionar uma carta de poder no campo de batalha ou utilizar efeitos. Após ambos os jogadores encerrarem seus turnos, a partida entra na fase de resolução de batalha.

Na fase de resolução de batalha, o jogo verifica as Cartas de Poder posicionadas por ambos jogadores em seus respectivos campos de batalha, levando em conta vantagens ou

desvantagens elementais e aumentos de poder. A carta com maior poder final vence a batalha, e o dano direto dessa carta é direcionado ao perdedor da fase. O jogo prossegue até que um dos jogadores reduza os pontos de vida do adversário a zero, ou que algum dos jogadores não tenha mais cartas disponíveis para comprar.

A figura 3.2.1 apresenta um diagrama que ilustra o fluxo de uma partida do jogo. Este diagrama serve como uma representação visual do andamento das etapas do jogo, permitindo uma compreensão mais clara das interações entre os jogadores e as ações disponíveis durante a partida.

Figura 3.2.1: Diagrama de fluxo do jogo



Fonte: Autor (2024)

De acordo com a figura 3.2.1, o fluxo do jogo se inicia com a opção de “Novo Jogo”, que leva ao estado “Início Batalha”. Neste momento, a batalha é preparada, e os jogadores recebem seus recursos iniciais, como cartas, *mana* e pontos de vida.

A partir do estado inicial, o jogo avança alternando entre o “Turno do Jogador” e o “Turno do Oponente”. Durante esses turnos, os jogadores realizam suas ações, como jogar cartas de poder ou efeito, respeitando as restrições explicadas anteriormente. Cada turno se encerra quando o jogador conclui suas ações, passando o controle para o próximo jogador.

Ao término do turno do oponente, o jogo entra no estado de “Resolução Batalha”, onde as cartas de poder posicionadas no campo de batalha são comparadas. Nesta etapa, variáveis importantes como a vida dos jogadores são atualizados conforme o resultado.

Após a resolução da batalha, o jogo verifica se alguma condição de vitória foi atingida, caso nenhum jogador tenha cumprido essa condição, o fluxo retorna ao estado de “Turno do Jogador”, e a partida continua normalmente. Quando alguma condição de vitória for satisfeita, o jogo é finalizado atingindo o estado “Fim”, onde é declarado o vencedor da batalha.

3.3. Cartas

As cartas são o núcleo da dinâmica do jogo e servem como principal ferramenta de interação dos jogadores durante a partida. Cada jogador possui um *deck* (conjunto de cartas), que contém uma variedade de cartas com diferentes funções e características. As cartas são categorizadas em dois tipos: **Cartas de Poder** e **Cartas de Efeito**. Cada tipo desempenha um papel específico na estratégia do jogo, contribuindo para a diversidade de jogadas e táticas que podem ser usadas ao longo das partidas.

Além de suas funções específicas, todas as cartas possuem um custo de *mana*, representado por um valor numérico necessário para que o jogador possa utilizar a carta durante seu turno. Essa mecânica proporciona aos jogadores um desafio do gerenciamento de seus recursos para poder utilizar as cartas desejadas, além de contribuir diretamente para o balanceamento do jogo, garantindo que cartas com o alto valor de poder ou efeitos poderosos demandem um maior gasto de recursos.

3.3.1. Cartas de Poder

As Cartas de Poder são essenciais para o combate direto entre os jogadores, uma vez que são responsáveis por causar dano. Cada carta de poder possui quatro atributos principais: custo de mana, poder, dano direto e elemento.

O custo de mana, como vimos anteriormente, determina a quantidade de recursos necessários para jogar a carta durante o turno. Já o atributo poder representa a força bruta da carta durante as batalhas, sendo um dos principais fatores para determinar o vencedor em uma interação direta.

Além do poder, as cartas possuem o atributo dano direto, que define o valor de dano aplicado diretamente aos pontos de vida do oponente quando a carta vence uma interação durante a fase de resolução de batalha.

Por fim, outro fator fundamental no desempenho das Cartas de Poder é o elemento associado a elas. Os elementos disponíveis no jogo são: FOGO, ÁGUA e GRAMA, onde cada um possui características únicas que influenciam as interações entre as cartas. O quadro 3.3.1.1 ilustra as interações de vantagens e desvantagens de cada elemento em relação aos outros:

Quadro 3.3.1.1: Vantagens e desvantagens

VANT / DESV	FOG	ÁGU	GRA
FOGO	0	-2	+2
ÁGUA	+2	0	-2
GRAMA	-2	+2	0

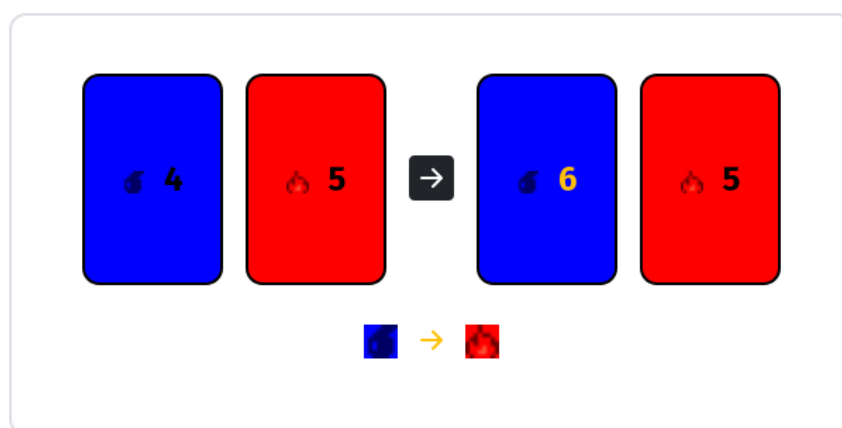
Fonte: Autor (2024)

Os valores presentes nas células do quadro 3.3.1.1 de vantagens e desvantagens representam um valor somado à força da carta de poder durante a interação dos elementos. Essa mecânica adiciona uma camada de estratégia ao jogo, pois os jogadores devem considerar não apenas a força bruta de suas cartas, mas também como os elementos interagem entre si.

Como exemplo, vamos considerar a interação entre cartas dos elementos FOGO e ÁGUA. Suponha que o “Jogador 1” jogue uma carta de poder 4 do elemento ÁGUA, enquanto o “Jogador 2” jogue uma carta do elemento FOGO com 5 de poder.

Segundo o quadro, a interação entre ÁGUA e FOGO resulta em uma penalidade de +2 para a carta de FOGO, já que ÁGUA tem vantagem sobre o elemento FOGO. Portanto, a interação resultará no quadro abaixo:

Quadro 3.3.1.2: Exemplo Interação ÁGUA-FOGO em Cartas de Poder.



Fonte: Autor (2024)

Visto que essa interação é aplicada somente uma vez a um dos jogadores, a força da carta de FOGO do “Jogador 2” permanece em 5, resultando na vitória do “Jogador 1” durante a rodada.

Esse sistema de interações elementares busca enriquecer a experiência do jogo, tornando cartas poderosas vulneráveis em determinadas situações, dependendo da interação entre os elementos.

3.3.2. Cartas de Efeito

As Cartas de Efeito desempenham um papel essencial na dinâmica do jogo, oferecendo aos jogadores diversas habilidades que podem alterar o fluxo da partida de maneira estratégica. Diferentemente das Cartas de Poder, que focam exclusivamente no combate direto, as Cartas de Efeito têm a capacidade de influenciar diversas mecânicas do jogo.

Essas cartas podem ser categorizadas em diferentes tipos, cada um com suas particularidades e impactos no jogo. Dentre seus atributos principais temos: tipo do efeito, valor do efeito e custo de mana.

O tipo de efeito é o atributo que define a natureza da habilidade de cada carta. Ele determina o que a carta pode fazer, como fortalecer cartas aliadas, restaurar pontos de vida do jogador ou comprar mais cartas.

Já o valor do efeito representa a magnitude ou intensidade do efeito aplicado pela carta. Por exemplo, uma carta com efeito de *APRIMORAMENTO* pode conceder +2 de poder a uma carta aliada, enquanto uma carta com efeito de *CURA* pode restaurar 3 pontos de vida ao jogador.

No quadro 3.3.1.2, descrevemos os principais tipos de efeitos disponíveis e suas respectivas funções:

Quadro 3.3.1.2: Exemplo de Cartas de Batalha em campo aplicando vantagens/desvantagens.

TIPO	DESCRIÇÃO
RECOMPRA	Descarta x cartas de sua mão e compra y cartas do Deck.
COMPRA	Compra x cartas do Deck para sua mão.
APRIMOR.	Aumenta o poder da carta em campo em x de poder.
CURA	Recupera x pontos de vida.

Fonte: Autor (2024)

Esses tipos de cartas permitem que os jogadores criem estratégias variadas, adaptando suas jogadas às circunstâncias da partida. Por exemplo, ao usar uma carta de efeito *COMPRA*, obtemos uma gama maior de cartas disponíveis, enquanto o outro jogador em sua rodada pode utilizar uma carta de *CURA* para recuperar sua vida durante a partida.

4. Implementação

Esta seção tem como objetivo detalhar o desenvolvimento do projeto proposto, apresentando as ferramentas e tecnologias utilizadas, a estrutura do sistema, e as estratégias aplicadas para resolver o problema proposto. Além disso, serão discutidos os desafios enfrentados durante a etapa de desenvolvimento. O jogo pode ser acessado através da plataforma github, no seguinte repositório: github.com/dasdwqdf/cgpl.

4.1. Ferramentas Utilizadas

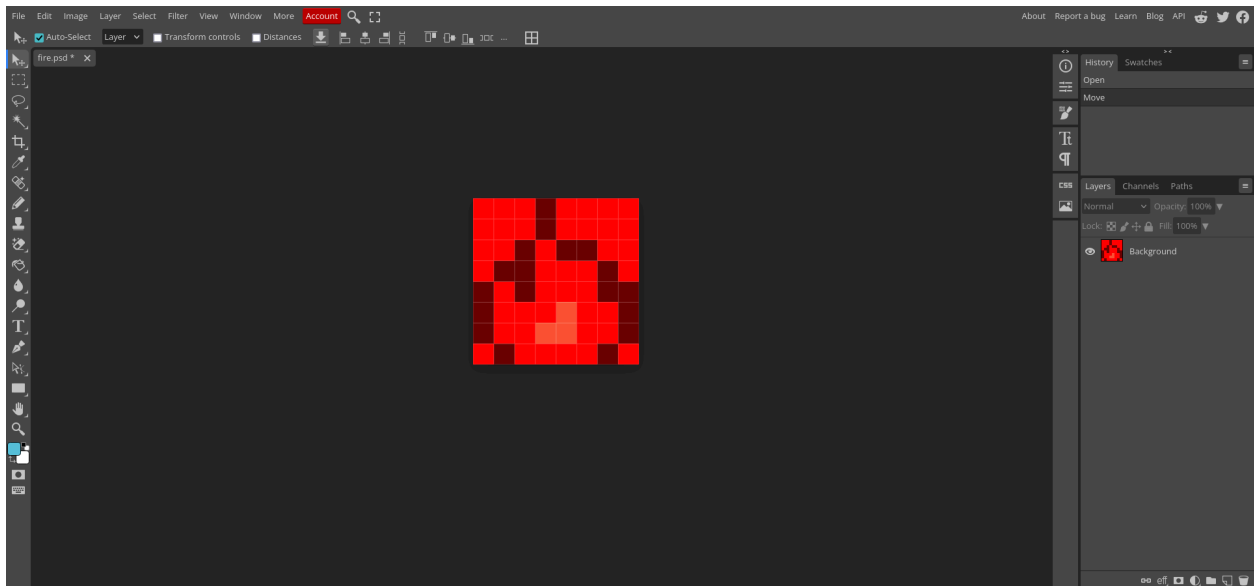
Para o desenvolvimento deste projeto, foram escolhidas ferramentas e tecnologias que se mostraram mais adequadas tanto às necessidades do jogo quanto ao nível de familiaridade com as ferramentas. Como linguagem de programação, foi escolhida a linguagem Java na versão 17, que, embora não tenha sido a linguagem originalmente planejada, foi adotada devido a experiência prévia do autor com ela. O Java oferece uma plataforma robusta, com portabilidade e desempenho suficientes para o tipo de jogo proposto.

Em relação às interfaces gráficas, foi optado pela biblioteca AWT (*Abstract Window Toolkit*), a qual é integrada juntamente com o JDK (*Java Development Kit*). Embora existam opções mais modernas e sofisticadas, o AWT foi escolhido pela simplicidade e porque se encaixava nas necessidades do projeto. Para a parte visual do jogo, foram criados *sprites* — imagens ou sequências de imagens utilizadas para representar personagens, objetos ou outros elementos dentro do jogo. Para a criação desses *sprites*, foi utilizada a ferramenta Photopea, uma plataforma de edição de imagens de fácil acesso e intuitiva.

Por fim, como ambiente de desenvolvimento, foi utilizada a *IDE* IntelliJ IDEA, da JetBrains, uma das ferramentas mais populares e completas para o desenvolvimento em Java nos dias atuais. Essa *IDE* oferece recursos que facilitam a codificação, gerenciamento, refatoração e organização do código.

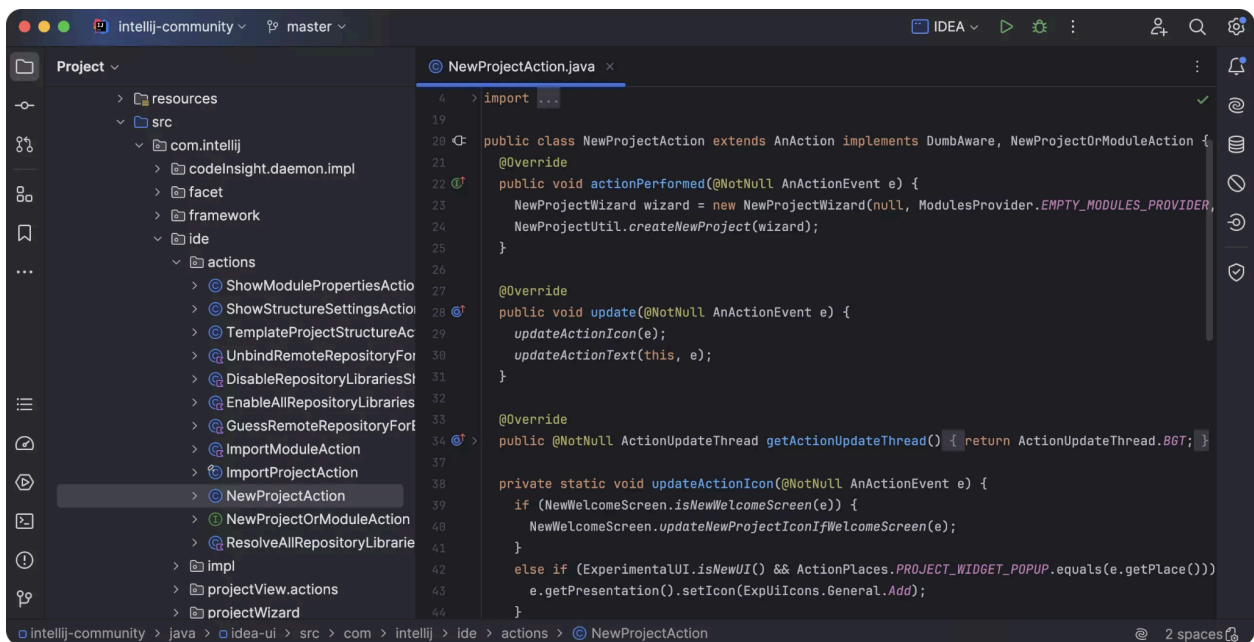
Segue abaixo algumas imagens das plataformas utilizadas para o desenvolvimento do jogo:

Figura 4.1.1: Ambiente de Edição de Imagens Photopea



Fonte: [Photopea](https://www.photopea.com/). Acesso em: 17 de dezembro de 2024.

Figura 4.1.2: IDE IntelliJ IDEA



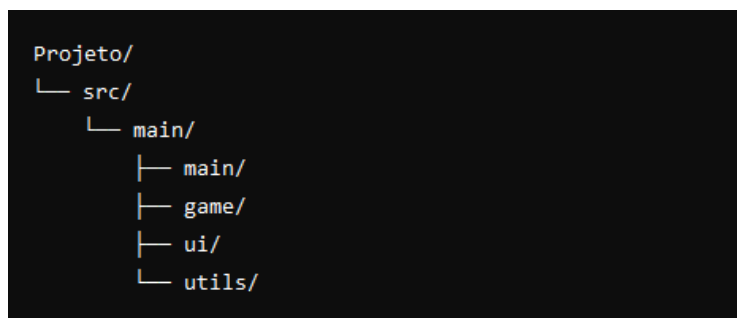
Fonte: [IntelliJ IDEA](https://www.jetbrains.com/idea/). Acesso em: 17 de dezembro de 2024.

4.2. Estrutura do Projeto

Devido à escolha da linguagem Java, o projeto foi desenvolvido seguindo o paradigma de orientação a objetos. Essa abordagem facilita a organização e reutilização do código, fazendo com que as funcionalidades sejam divididas em classes e métodos. Com base na seção 3, onde foi abordado o fluxo geral do jogo e sua mecânica, nesta subseção exploraremos as principais classes, como elas foram gerenciadas e a estrutura do projeto.

O projeto foi estruturado de forma modular, visando a separação das responsabilidades entre os diferentes componentes do jogo. De forma geral, a estrutura do projeto é composta por quatro pacotes principais, sendo eles (quadro 4.2.1):

Quadro 4.2.1: Estrutura do Projeto



Fonte: Autor (2024)

Primeiramente, o pacote `Main` contém a classe responsável pela inicialização do jogo. Nele está localizada a função principal, onde o jogo é iniciado e as configurações iniciais são realizadas. A classe `Main` é o ponto de entrada do sistema, a partir da qual as demais funcionalidades são iniciadas.

O pacote `Game` abrange as principais classes relacionadas à lógica do jogo. Aqui são definidas as regras do jogo, as mecânicas e a inteligência artificial. Este pacote é essencial, pois define a parte interna e o funcionamento do jogo.

Em seguida, o pacote `UI` é responsável pela interface gráfica do jogo. Ele contém as classes que definem as telas e os componentes visuais, como menus, botões e janelas do jogo. A interação com o usuário é gerenciada por este pacote, onde ele se comunica com a parte interna do jogo.

Por fim, o pacote `Utils` reúne funções auxiliares que são utilizadas em diversas partes do projeto. As classes localizadas neste pacote não estão diretamente relacionadas à lógica do jogo, mas oferecem utilitários importantes, facilitando a reutilização de código e minimizando a duplicação de funcionalidades.

4.3. Lógica do Jogo e Suas Classes

Como vimos anteriormente, o pacote `Game` abriga as principais classes responsáveis pela lógica do jogo. Nas próximas subseções, abordaremos essas classes de maneira geral, detalhando suas responsabilidades e como elas contribuem para o funcionamento interno do jogo. Dentre as classes mais importantes relacionadas a funcionalidade do jogo temos `Card`, `Battle` e `PlayerEntity`.

4.3.1. Card

A classe `Card` é responsável por representar as cartas utilizadas durante a partida. Ela define características de cada carta, como seu valor, tipo e efeitos. A seguir, apresentamos um trecho simplificado do código da classe `Card` (quadro 4.3.1.1):

Quadro 4.3.1.1: Trecho de Código simplificado para Classe `Card`

```
public class Card {  
    String name;  
    int manaCost;  
    CardType cardType;  
    String description;  
    CardElement cardElement;  
    int power;  
    int directDamage;  
    CardEffect cardEffect;  
    int effectArg;  
}
```

Fonte: Autor (2024)

No **Quadro 4.3.1.1**, visualizamos que a classe `Card` possui quatro atributos principais: `name`, `manaCost`, `cardType` e `description`, os quais representam características gerais de uma carta.

A partir do atributo `cardType`, derivam-se dois construtores responsáveis por preencher determinados atributos da classe. Cada construtor é projetado para lidar com características distintas, dependendo do tipo da carta.

Por exemplo, ao determinar-se que `cardType` é do tipo `POWER`, os atributos relacionados a cartas de poder, como `directDamage`, `cardElement` e `power`, são inicializados. Já quando o `cardType` é definido como `EFFECT`, o foco passa a ser a aplicação de efeitos especiais durante a partida. Nesse sentido, os atributos `cardEffect` e `effectArg` são preenchidos.

Seguindo a estrutura de gerenciamento das cartas, temos a classe `CardManager`, responsável por operações como embaralhamento das cartas, importação de baralhos, etc. Essa classe centraliza o controle sobre o conjunto de cartas e facilita a execução de mecânicas durante o jogo.

A seguir, apresentamos uma versão simplificada da classe `CardManager` (quadro 4.3.1.2):

Quadro 4.3.1.2: Trecho de Código simplificado para Classe `CardManager`

```
public class CardManager {  
    List<Card> deck;  
    List<Card> hand;  
    List<Card> discard;  
  
    public void initDeck(String deckName);  
    public void shuffleDeck();  
    public int drawCards(int numCards);  
    public int discardCards(int numCards);  
    public void useCard(Card card);  
}
```

Fonte: Autor (2024)

No **Quadro 4.3.1.2**, observamos que a classe `CardManager` organiza as cartas em três listas distintas, facilitando a separação lógica entre as cartas compradas, em uso e descartadas.

O método `initDeck(String deckName)` é responsável por carregar as cartas de um baralho específico através de um `CardParser`. Já o método `shuffleDeck()` embaralha o baralho atual, garantindo aleatoriedade na compra de cartas. Por outro lado, a função `drawCards(int numCards)` adiciona um número específico de cartas à mão do jogador, enquanto `discardCards(int numCards)` remove as cartas da mão e as transfere para a pilha de descarte. Por fim, da mesma forma o método `useCard(Card card)` move uma carta para a pilha de descarte.

4.3.2. PlayerEntity

A classe `PlayerEntity` desempenha um papel central no gerenciamento das entidades do jogo que possuem interações diretas com as cartas e recursos. Sendo uma classe abstrata, a qual serve como base para as classes derivadas que representam o jogador controlável e a inteligência artificial. Ela define os atributos essenciais que os jogadores compartilham, como por exemplo a quantidade máxima de *mana*, o nome do jogador, pontos de vida, *mana* atual e a carta em campo.

A partir da classe `PlayerEntity`, surgem duas subclasses: `ControllableEntity` e `AiEntity`. A `ControllableEntity` representa o jogador controlado por uma pessoa, ou seja, o jogador humano. Por outro lado, a `AiEntity` representa o jogador controlado pela inteligência artificial. Assim como a `ControllableEntity`, a `AiEntity` herda os atributos e comportamentos da `PlayerEntity`, mas suas ações são determinadas por regras ou algoritmos de decisão, os quais serão abordados em seções futuras, ao invés de serem controladas por um ser humano.

A seguir, apresentamos um trecho de código que ilustra a estrutura da classe `PlayerEntity` (quadro 4.3.2.1):

Quadro 4.3.2.1: Trecho de Código simplificado para Classe `PlayerEntity`


```
public abstract class PlayerEntity {  
    public static int maxMana = 3;  
    String name;  
    int hp;  
    int mana;  
    CardManager cardManager;  
    Card fieldCard;  
}
```

Fonte: Autor (2024)

4.3.3. Battle

A classe `Battle` controla as interações entre os jogadores durante o jogo. Ela gerencia o estado da batalha, o turno dos jogadores e a execução das fases de combate. É responsável também por definir o andamento da partida e coordenar a troca de ações entre os jogadores, garantindo que as regras do jogo sejam seguidas corretamente. Entre seus atributos, o `battleState` define o estado atual da batalha (se está em andamento, pausada ou finalizada), enquanto `players` mantém a lista de jogadores. Além desses atributos, temos `currentTurn` e `currentPlayerIndex`, que são variáveis de controle do fluxo da partida.

Dentro dessa estrutura, a classe `Battle` depende de três *handlers* essenciais para organizar e coordenar as diferentes fases da partida. O `BattleMessageHandler` é responsável por controlar o fluxo de mensagens, garantindo a comunicação das ações realizadas durante a partida com as telas do jogo.

Em seguida, o `CombatPhaseHandler` gerencia as fases de resolução de combate, aplicando as regras de combate apresentadas na seção 3. Ele é responsável por realizar a verificação e a aplicação das vantagens elementais, calcular danos, e definir o vencedor de cada fase de combate.

Por fim, o `EffectHandler` lida com a aplicação dos efeitos das cartas e habilidades dos jogadores. Ele é encarregado de aplicar os efeitos específicos que podem alterar os status dos jogadores ou das cartas, como curas, aprimoramentos ou recuperação de *mana*.

O trecho de código a seguir ilustra a classe `Battle` juntamente com os atributos mencionados anteriormente (quadro 4.3.3.1):

Quadro 4.3.3.1: Trecho de Código simplificado para Classe **Battle**

```
public class Battle {  
    BattleState battleState;  
    List<PlayerEntity> players;  
  
    int currentTurn;  
    int currentPlayerIndex;  
  
    BattleMessageHandler battleMessageHandler;  
    CombatPhaseHandler combatPhaseHandler;  
    EffectHandler effectHandler;  
}
```

Fonte: Autor (2024)

Em termos de pseudocódigo, o fluxo da batalha se resumiria da seguinte forma: o jogo inicia com a configuração da batalha, onde o `battleState` é definido como “em andamento”, a lista de jogadores é inicializada, e os turnos são controlados através das variáveis de controle. Após a finalização do turno de ambos jogadores, a classe `CombatPhaseHandler` determina o vencedor da rodada, atualizando o status de ambos os jogadores. Em seguida, o sistema verifica se a partida foi finalizada, isto é, caso algum dos jogadores tenha cumprido uma condição de vitória. Caso a batalha não tenha sido finalizada, o fluxo reinicia-se até que o fim da batalha seja atingido.

A estrutura apresentada nesta seção detalhou os principais componentes responsáveis pela lógica central do jogo, desde a manipulação de cartas até o gerenciamento dos jogadores e condução das batalhas. Na próxima seção, exploraremos como os *sprites* e os elementos gráficos foram implementados para dar vida ao jogo, tornando ele mais atraente para o jogador visualmente.

4.4. Sprites



Os *sprites* são componentes importantes para a representação visual dos elementos em jogos do estilo 2D, como objetos interativos, cenários, personagens, entre outros. Eles são

imagens ou sequências de imagens que representam objetos do jogo, permitindo uma interação visual mais intuitiva e atrativa entre o jogador e o ambiente.

No contexto do desenvolvimento dos *sprites*, eles seguiram uma estética retrô inspirada nos clássicos de 8, 16 e 32 bits. A seguir, são apresentadas as principais categorias de sprites utilizados no jogo, juntamente com uma breve descrição de suas funções e o contexto em que são aplicados.

O primeiro quadro apresenta os ícones associados aos tipos de cartas disponíveis no jogo, destacando suas funções e características visuais.




Quadro 4.4.1: Sprites de Tipo de Cartas

SPRITE	NOME	DESCRIÇÃO
	POWER	Carta do Tipo Poder
	EFFECT	Carta do Tipo Efeito

Fonte: Autor (2024)

No **Quadro 4.4.1**, o ícone *POWER* representa as cartas de poder, que são voltadas para ações ofensivas, destacando-se pelo tom de vermelho e a imagem de um punho. Já o ícone *EFFECT* identifica cartas de efeito com habilidades especiais, sendo utilizado o tom roxo.








Quadro 4.4.2: Sprites de Tipo de Elementos

SPRITE	NOME	DESCRIÇÃO
	FIRE	Elemento Fogo
	WATER	Elemento Água
	GRASS	Elemento Grama

Fonte: Autor (2024)

O **Quadro 4.4.2** apresenta os ícones que representam os elementos dentro do jogo, utilizados para definir as características e afinidades das cartas. No quadro, o ícone *FIRE* simboliza o fogo, *WATER* representa o elemento água e *GRASS* ilustra o elemento grama.

Quadro 4.4.3: Sprites de *Status*

SPRITE	NOME	DESCRIÇÃO
	EMPTY-MANA	Mana Vazia
	FILLED-MANA	Mana Cheia
	HEART	Vida
	DIRECT-DAMAGE	Dano Direto
	POWER-UP	Aprimoramento
	ELEMENTAL-POWER-UP	Aprimoramento Elemental
	ELEMENTAL-POWER-DOWN	Redução Elemental

Fonte: Autor (2024)









Os *sprites* apresentados no **Quadro 4.4.3** representam efeitos ou condições associados às cartas ou aos jogadores durante a partida. Cada um desses ícones busca transmitir as informações de forma clara e rápida sobre o estado atual do jogo.

O ícone *EMPTY-MANA* simboliza a ausência de *mana*, indicando que o jogador não possui energia para realizar alguma ação. Em contrapartida, *FILLED-MANA* ilustra a presença de mana, consequentemente a possibilidade para realizar uma ação, o *sprite FILLED-MANA* também é utilizado para indicar o custo de mana necessário para a utilização de uma carta.

Além dos *sprites* de *mana*, o ícone *HEART* representa a vida do jogador, refletindo sua saúde atual durante a partida. O *DIRECT-DAMAGE* representa o atributo dano direto das cartas de poder, geralmente visível onde temos informações sobre as cartas. Por outro lado, o ícone *POWER-UP* simboliza um aumento temporário de poder, representando um bônus utilizado pelo jogador.

Por fim, os ícones *ELEMENTAL-POWER-UP* e *ELEMENTAL-POWER-DOWN* representam, respectivamente, o aumento e redução de poder devido às interações elementares, destacando mudanças de poder nas cartas dos jogadores.

Quadro 4.4.4: Sprites de Personagens

SPRITE	NOME	DESCRIÇÃO
	PLAYER-IDLE	Jogador Parado
	PLAYER-HEAL	Jogador recuperando Vida
	PLAYER-MANA	Jogador recuperando Mana
	PLAYER-DAMAGE	Jogador recebendo Dano
	BOT-IDLE	Bot Parado
	BOT-HEAL	Bot recuperando Vida
	BOT-MANA	Bot recuperando Mana
	BOT-DAMAGE	Bot recebendo Dano

Fonte: Autor (2024)

A partir do **Quadro 4.4.4**, são apresentados os sprites relacionados aos personagens do jogo. O *sprite* *PLAYER-IDLE* é utilizado para representar o estado ocioso do personagem, quando não está realizando nenhuma ação específica. *PLAYER-HEAL* simboliza o momento em que o personagem está realizando um processo de cura, através de uma carta de cura. Já o ícone *PLAYER-MANA* representa o estado em que o personagem recupera mana, refletindo condições para realizar ações com a mana recuperada.

Por fim, *PLAYER-DAMAGE* é o *sprite* que aparece quando o personagem recebe dano, provido da fase de combate. Da mesma forma, para os *NPCs*, o quadro apresenta os *sprites* *BOT-IDLE*, *BOT-HEAL*, *BOT-MANA* e *BOT-DAMAGE*, com a mesma funcionalidade dos ícones do jogador, mas aplicados ao *bot*.

Compreender a utilização de sprites no jogo permite uma visão mais clara sobre como os elementos visuais interagem com o jogador e como cada ação é representada. Assim como os *sprites* têm um papel fundamental na representação visual das ações e status, as telas do

jogo desempenham um papel igualmente importante na organização e apresentação dessas informações ao jogador. Na seção subsequente, será explorada a disposição das telas, a estrutura de cada tela e seus componentes.

4.5. Telas

As telas do jogos são responsáveis por organizar e apresentar as informações de forma visual e interativa para o jogador. Cada tela é gerenciada por um controlador, responsável por lidar com as atualizações de estado e a renderização dos elementos gráficos.

No contexto de desenvolvimento de jogos, podemos compreender um jogo como um loop infinito, que é responsável por manter o fluxo contínuo de execução. Esse loop é dividido em duas funções principais: `update()` e `draw()`.

A função `update()` é encarregada de processar as entradas do jogador, calcular a lógica do jogo e atualizar os estados das variáveis e objetos na tela. Por exemplo, ela verifica se uma tecla foi pressionada, ativando uma ação dentro da lógica do jogo.

Já a função `draw()` é responsável por renderizar os gráficos na tela com base no estado atualizado. Isso inclui desenhar personagens, cartas, vida do personagem e qualquer outro elemento visual relevante, garantindo que o jogador veja as mudanças em tempo real.

Para o jogo desenvolvido, a estrutura abordada é aplicada nas duas principais telas: a **Tela Inicial** e a **Tela de Batalha**. Cada uma dessas telas possui um propósito específico e é projetada para oferecer funcionalidades distintas.

Quadro 4.5.1: Tela Inicial



Fonte: Autor (2024)

A Tela Inicial apresentada no **Quadro 4.5.1**, é o ponto de entrada do jogo e serve como interface principal para o jogador iniciar a partida. Nela, são exibidos o título do jogo e as opções de menu: “Jogar” e “Sair”.

A navegação entre as opções é feita por meio das teclas direcionais, e a seleção atual é indicada por um marcador posicionado ao lado do texto. Para confirmar a seleção e iniciar a ação desejada, basta pressionar a tecla “X”.

Ao selecionar a ação de “Jogar” pelo menu inicial, a Tela de Batalha é exibida. Nessa tela, ocorre a interação entre o jogador e o jogo, que é organizada em três componentes principais: a caixa de texto de batalha, o painel do jogador e o painel de turno, conforme o quadro abaixo:

Quadro 4.5.2: Tela de Batalha



Fonte: Autor (2024)

O **Quadro 4.5.2** apresenta uma visão geral da Tela de Batalha e seus principais componentes, a seguir discutiremos cada um desses componentes individualmente, destacando suas funções e interações.

Quadro 4.5.3: Painel de Turno



Fonte: Autor (2024)

O **Painel de Turno**, mostrado no **Quadro 4.5.3**, é um componente ilustrativo que exibe informações relacionadas ao andamento da batalha, mantendo o jogador informado sobre a sequência temporal do jogo.

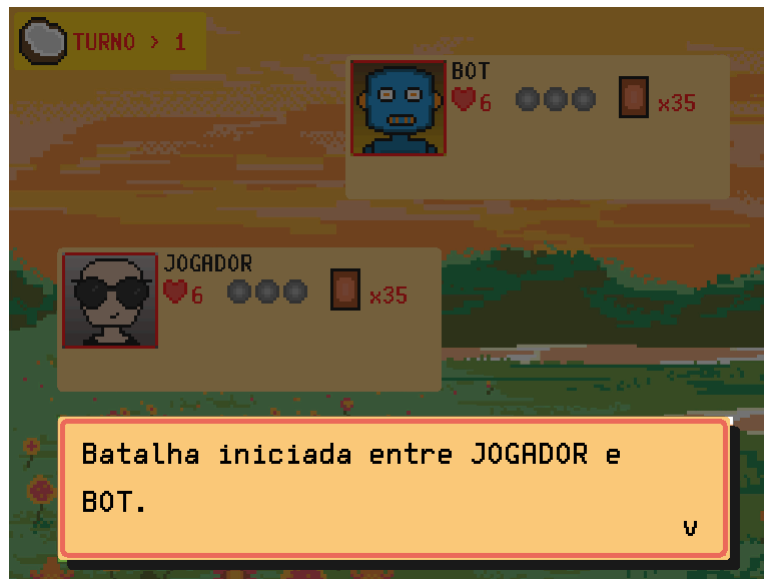
Quadro 4.5.4: Painel do Jogador



Fonte: Autor (2024)

O **Painel do Jogador**, conforme ilustrado no **Quadro 4.5.4**, contém informações sobre o estado atual do jogador, como pontos de vida, recursos disponíveis como *mana*, número de cartas no baralho, entre outros. Além disso, é neste painel que ocorrem animações que indicam efeitos importantes durante a batalha, como a recuperação de vida e mana, bem como a ação de colocar cartas em campo e levar dano. Essas animações fornecem um *feedback* visual, auxiliando o jogador a acompanhar as mudanças do estado do jogo em tempo real.

Quadro 4.5.5: Caixa de Texto de Batalha



Fonte: Autor (2024)

A **Caixa de Texto de Batalha**, conforme destacada pelo **Quadro 4.5.5** é um componente, que exibe as mensagens das ações que acontecem durante a batalha, como utilização de cartas, atualização de *status* dos jogadores, resultados de ataques, etc. Esta caixa fornece o *feedback* necessário para que o jogador compreenda o que está acontecendo durante a batalha.

Além das mensagens de batalhas, a **Caixa de Texto de Batalha** possui outros dois modos de operação: um onde o jogador seleciona as opções de ação durante o turno, e outro dedicado à seleção de cartas a serem usadas.

Quadro 4.5.6: Caixa de Texto de Batalha no modo de Seleção de Ação



Fonte: Autor (2024)

O **Quadro 4.5.6** apresenta a **Caixa de Texto de Batalha** no **modo de seleção de ação**, onde o jogador tem duas opções principais: “MÃO” e “ENCERRAR TURNO”. A opção atualmente selecionada é indicada por um marcador, que é posicionado ao lado do texto correspondente. Para confirmar a seleção, o jogador deve pressionar a tecla “x”.

Durante esse modo, a seleção pode ser alterada utilizando as teclas direcionais do teclado. As opções são alternadas com o auxílio dos direcionais do teclado, permitindo que o jogador escolha entre visualizar sua mão de cartas ou encerrar o turno.

Ao confirmar a opção “MÃO”, a **Caixa de Texto de Batalha** muda para o modo de **seleção de cartas**, durante esse modo, um novo painel, chamado **Painel da Carta**, é posicionado à direita do **Painel do Jogador**. Nesse painel, o jogador consegue verificar informações detalhadas referente a carta selecionada, como custo de *mana*, elemento, dano direto e sua descrição. Cada carta é exibida nesse painel conforme é selecionada pela **Caixa de Texto**.

Assim como nos outros painéis, **Caixa de Texto de Batalha** neste modo de **seleção de cartas** é navegável utilizando as teclas direcionais. Para confirmar a utilização de uma carta, o jogador deve pressionar a tecla “x”. Uma diferenciação importante desse modo é que,

caso o jogador deseje voltar ao **modo de seleção de ação**, basta pressionar a tecla “z”, retornando à tela anterior e permitindo que a opção “ENCERRAR TURNO” seja selecionada.

O **Quadro 4.5.7** abaixo ilustra os detalhes do **modo de seleção de cartas** na **Caixa de Texto de Batalha**, mostrando a disposição das cartas disponíveis, a navegação entre as cartas e o comportamento visual da seleção de uma carta.

Quadro 4.5.7: Caixa de Texto de Batalha no modo de Seleção de Carta



Fonte: Autor (2024)

Após discutirmos os principais componentes visuais da interface, que desempenham um papel fundamental na interação do jogador com o jogo, a próxima seção abordará um aspecto igualmente importante: o som. Embora de forma mais superficial, exploraremos como os efeitos sonoros foram integrados ao jogo, incluindo a escolha e implementação dos sons para ações, efeitos de ambiente e *feedbacks* em geral.

4.6. Áudio

Para gerenciar os áudios no jogo, foi criada uma classe chamada `Sound`, responsável pela execução e controle das músicas e efeitos sonoros. As músicas de fundo, que acompanham as telas do jogos, tocando em *loop* infinito enquanto o jogador estiver presente na respectiva tela. As músicas utilizadas são de uso livre disponibilizadas pelo **Fesliyan Studios**.

Em relação aos efeitos sonoros, estes foram gerados utilizando a ferramenta **jsfxr**. Através dessa ferramenta, os efeitos foram ajustados por meio de alterações nos parâmetros da ferramenta, até que os resultados fossem satisfatórios para o contexto do jogo. Além disso, o som foi integrado à mecânica do jogo, com cada ação realizada pelas entidades jogadores (como ataques, curas ou uso de aprimoramentos) sendo acompanhada por efeitos sonoros específicos, garantindo um *feedback* auditivo.

O quadro a seguir apresenta a interface da ferramenta **jsfxr** (quadro 4.6.1), utilizada para criação de efeitos sonoros.

Quadro 4.6.1: jsfxr - 8 bit sound maker and online sfx generator

The image shows the jsfxr web interface, which is divided into three main sections: Generator, Manual Settings, and Sound. The Generator section on the left lists various sound effects like Pickup/coin, Laser/shoot, Explosion, Powerup, Hit/hurt, Jump, Click, Blip/select, Synth, Tone, Mutate, and Play. The Manual Settings section in the center allows for detailed configuration of these sounds, including Envelope (Attack, Sustain, Decay times), Frequency (Start frequency, Min freq. cutoff, Slide, Delta slide), Vibrato (Depth, Speed), Arpeggiation (Frequency mult, Change speed), Duty Cycle (Duty cycle, Sweep), Retrigger (Rate), and Flanger (Offset, Sweep). The Sound section on the right provides download links, file size (14kB), sample count (14666), gain (-10.93 dB), sample rate (44k, 22k, 11k, 6k), and sample size (16 bit, 8 bit) options. It also includes a permalink and a copy code button.

Fonte: [jsfxr - 8 bit sound maker and online sfx generator](https://jsfxr.com/) Acesso em: 22 de dezembro de 2024.

Após o detalhamento dos aspectos sonoros do jogo, é importante entendermos como as informações são atualizadas dinamicamente na tela para refletir o andamento da batalha. Dessa forma, na próxima seção, abordaremos em detalhes como essas atualizações ocorrem, garantindo que o jogador tenha sempre uma visão clara e precisa do estado atual do jogo, com *feedback* contínuo sobre as ações e eventos da partida.

4.7. Interação Tela-Jogo

Para gerenciar as atualizações de mensagens e informações no jogo, foi criada a classe `BattleMessageHandler`, que desempenha um papel fundamental na comunicação entre o jogo e a interface visual. As mensagens geradas durante a batalha, como ataques, curas e mudanças de status, são armazenadas em uma fila e consumidas à medida que o jogador avança pela **Caixa de Texto de Batalha**, através do **controlador da Tela de Batalha**. Esse processo permite que o jogo forneça *feedback* ao jogador de forma sequencial e organizada.

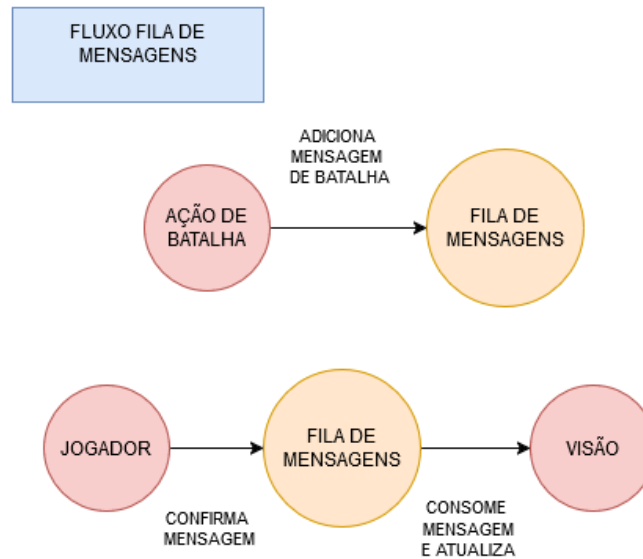
Cada mensagem adicionada à fila é representada por um objeto da classe `BattleMessage`, que contém o texto da mensagem e um *snapshot* do estado do jogador no momento em que a mensagem foi registrada. Esses *snapshots* são fundamentais, pois garantem que o estado do jogo seja atualizado de maneira sequencial e precisa. O *snapshot* inclui informações como os pontos de vida do jogador, recursos disponíveis e outros dados relevantes para o andamento da batalha.

A medida que essas mensagens são consumidas pelo `BattleMessageHandler`, vários efeitos são acionados. O texto da mensagem é exibido na **Caixa de Texto de Batalha**, oferecendo um *feedback* textual imediato. Além disso, efeitos sonoros e animações são disparados, dependendo do tipo de mensagem, como sons de ataque, cura e suas respectivas animações.

Outro ponto relevante é a atualização dos painéis de status dos jogadores. Com base nos *snapshots* associados a cada mensagem, as informações nos painéis são ajustadas para refletir as alterações no estado do jogo. Por exemplo, após um ataque, o painel de vida do jogador é atualizado para refletir a perda de pontos de vida.

Para facilitar a compreensão desse processo e ilustrar de forma mais clara como ele funciona, abaixo apresentamos um fluxograma (figura 4.7.1) que exemplifica visualmente o funcionamento do `BattleMessageHandler` e sua interação com o visual do jogo, incluindo animações, efeitos sonoros e atualização de valores nos componentes visuais.

Figura 4.7.1: Fluxograma Interação Tela-Jogo



Fonte: Autor (2024)

4.8. Abordagens de IA

No desenvolvimento do sistema de combate do jogo, foram implementadas duas estratégias distintas de inteligência artificial, ambas baseadas em um modelo de regras predefinidas. Essas abordagens foram projetadas para controlar o comportamento do adversário durante os turnos, aplicando diferentes prioridades para a tomada de decisão. A IA foi construída com base em **heurísticas** de tomada de decisão, onde o agente toma ações sequenciais com base em condições predefinidas que avaliam as situações de jogo.

As estratégias utilizam condições específicas para decidir quais ações realizar durante o turno, priorizando objetivos complementares: maximizar o impacto imediato ou conservar recursos para jogadas futuras. Ambas as IAs seguem uma lógica sequencial para selecionar cartas e administrar a *mana* disponível.

Antes das abordagens de IA se especializarem em seus objetivos específicos, elas passam por um processo de avaliação de condições críticas, como a necessidade de restaurar mana, curar pontos de vida ou comprar mais cartas. Em seguida, as regras específicas de cada abordagem são aplicadas para definir as jogadas a serem realizadas.

Nos tópicos a seguir, serão descritas detalhadamente as duas estratégias implementadas, destacando suas características e objetivos.

4.8.1. IA Focada na Maximização de Poder

A primeira abordagem foi desenvolvida para maximizar o impacto de cada turno, utilizando cartas que possuem os maiores efeitos disponíveis. Essa IA segue regras que priorizam o uso da mana em cartas de poder ou aprimoramento mais poderosas, com o objetivo de causar o maior dano possível ao jogador no menor tempo. Em termos de pseudocódigo, ela pode ser descrita como:

Quadro 4.8.1.1: Pseudocódigo primeira abordagem IA

```
if (cartasDePoderEmCampo == null) {
    ordenarCartasPoderPorForca();

    for (Carta carta : listaCartasPoder) {
        if (manaAtual >= carta.custoCarta) {
            movimentos.adicionar(carta);
            atualizarMana();
            break;
        }
    }
}

ordernarCartasAprimoramentoPorValor();

for (Carta carta : listaCartasAprimoramento) {
    if (manaAtual >= carta.custoCarta) {
        movimentos.adicionar(carta);
        atualizarMana();
    }
}

return movimentos;
```

Fonte: Autor (2024)

Essa abordagem não considera a necessidade de economizar recursos para turnos subsequentes, sendo ideal para cenários onde um impacto ofensivo imediato é vantajoso.

4.8.2. IA Focada na Conservação de Mana

A segunda abordagem prioriza o gerenciamento eficiente de recursos, mantendo um nível de mana disponível ao final de cada turno. O objetivo dessa estratégia é garantir a flexibilidade para futuras jogadas, evitando situações em que o adversário fique sem opções viáveis em turnos posteriores. No quadro abaixo, segue um pseudocódigo da lógica geral dessa abordagem:

Quadro 4.8.2.1: Pseudocódigo segunda abordagem IA

```
if (cartasDePoderEmCampo == null) {  
    ordenarCartasPoderPorCustoMana();  
  
    for (Carta carta : listaCartasPoder) {  
        if (manaAtual >= carta.custoCarta) {  
            movimentos.adicionar(carta);  
            atualizarMana();  
            break;  
        }  
    }  
}  
  
ordenarCartasAprimoramentoPorCustoMana();  
  
for (Carta carta : listaCartasAprimoramento) {  
    if (manaAtual >= carta.custoCarta) {  
        movimentos.adicionar(carta);  
        atualizarMana();  
    }  
}  
  
return movimentos;
```

Fonte: Autor (2024)

Após descrevermos as duas abordagens de IA, Focada na Maximização de Poder e Focada na Conservação de Recursos, passamos agora para a seção de análise dos resultados obtidos. Nesta próxima seção apresentaremos uma análise dos resultados, levando em

consideração tanto as medidas quantitativas quanto qualitativas. A análise será dividida em dois momentos: no primeiro momento, discutiremos a metodologia e as configurações utilizadas durante o teste, e, em seguida, serão apresentados os resultados obtidos.

5. Metodologia e Configurações

Para realizar a comparação entre as duas abordagens de IA, um total de 100 partidas foi disputado entre as duas IAs utilizando o mesmo *deck*, com a seguinte configuração:

Quadro 5.1: Cartas utilizadas no *deck* de testes

```
0, TSUNAMI, 2, WATER, 8, 2, CAUSA DANO DE ÁGUA AO Oponente
0, INFERNO, 3, FIRE, 10, 3, CAUSA DANO DE FOGO AO Oponente
0, FLORESTA, 3, GRASS, 12, 2, CAUSA DANO DE GRAMA AO Oponente
0, VORTEX, 2, WATER, 7, 1, CAUSA DANO DE ÁGUA AO Oponente
0, ERUPCAO, 1, FIRE, 4, 2, CAUSA DANO DE FOGO AO Oponente
0, RELVA, 1, GRASS, 5, 1, CAUSA DANO DE GRAMA AO Oponente
0, VORTEX, 2, WATER, 7, 1, CAUSA DANO DE ÁGUA AO Oponente
0, ERUPCAO, 1, FIRE, 4, 2, CAUSA DANO DE FOGO AO Oponente
0, RELVA, 1, GRASS, 5, 1, CAUSA DANO DE GRAMA AO Oponente
0, GELO, 1, WATER, 4, 1, CAUSA DANO DE ÁGUA AO Oponente
0, BRASA, 1, FIRE, 2, 1, CAUSA DANO DE FOGO AO Oponente
0, FOLHAS, 1, GRASS, 1, 2, CAUSA DANO DE GRAMA AO Oponente
0, GELO, 1, WATER, 4, 1, CAUSA DANO DE ÁGUA AO Oponente
0, BRASA, 1, FIRE, 2, 1, CAUSA DANO DE FOGO AO Oponente
0, FOLHAS, 1, GRASS, 1, 2, CAUSA DANO DE GRAMA AO Oponente
0, EXPLOSAO, 2, FIRE, 6, 4, CAUSA DANO DE FOGO AO Oponente
0, TORREN, 0, WATER, 1, 3, CAUSA DANO DE ÁGUA AO Oponente
0, TERRA, 0, GRASS, 2, 3, CAUSA DANO DE GRAMA AO Oponente
1, COMPRA, 0, DRAW, 1, COMPRE 1 CARTA
1, COMPRA, 0, DRAW, 1, COMPRE 1 CARTA
1, MANA+, 0, MANA, 1, REGENERA 1 DE MANA
1, MANA+, 0, MANA, 1, REGENERA 1 DE MANA
1, MANA+, 0, MANA, 1, REGENERA 1 DE MANA
1, SORTE, 1, REDRAW, 6, DESCARTE SUA MÃO E COMPRE 6 CARTAS
1, BENCAO, 1, BUFF, 1, AUMENTA O PODER EM 1
1, BENCAO, 1, BUFF, 1, AUMENTA O PODER EM 1
1, VIGOR, 2, BUFF, 2, AUMENTA O PODER EM 2
```

```
1,VIGOR,2,BUFF,2,AUMENTA O PODER EM 2  
1,VENTOS,1,HEAL,1,RECUPERA 1 DE VIDA  
1,CURAR,3,HEAL,2,RECUPERA 2 DE VIDA
```

Fonte: Autor (2024)

O *deck* é composto por cartas de diferentes tipos e efeitos, incluindo cartas de dano (água, fogo e grama), cura, aprimoramento de poder e regeneração de mana. A construção do deck foi feita de forma a garantir um equilíbrio entre as diversas mecânicas do jogo, considerando que cartas mais poderosas exigem um custo maior de mana, enquanto cartas mais fracas exigem um custo menor. Essa diversidade de cartas permite que as IAs explorem diferentes estratégias e abordagens de jogo.

Todas as partidas foram realizadas sob as mesmas condições, ou seja, as IAs enfrentaram-se utilizando o mesmo deck, de modo a garantir que as diferenças de desempenho fossem atribuídas às estratégias adotadas pelas abordagens e um pouco de sorte durante o embaralhamento e compra das cartas.

Essas condições de teste visam fornecer uma análise justa e equilibrada do impacto das diferentes estratégias de IA e suas performances durante as partidas, permitindo uma comparação objetiva dos resultados.

6. Análise dos Resultados

Após a realização da simulação de 100 partidas entre as duas IAs, com a utilização do mesmo deck em todas as partidas, foram coletados diversos dados sobre o desempenho de cada abordagem. A seguir, apresentaremos uma análise detalhada dos resultados, incluindo as principais medidas observadas e as implicações desses resultados.

6.1. Resultados Médios

Os resultados médios obtidos ao longo das 100 partidas indicam as diferenças no desempenho entre as duas abordagens de IA: a **IA Focada na Maximização de Poder** e a **IA Focada na Conservação de Mana**.

Para avaliar o desempenho de cada abordagem, foram utilizadas quatro medidas principais, sendo elas: poder total acumulado, número médio de movimento por batalha, mana não utilizada e número de vitórias.

A medida poder total acumulado refere-se à soma de poder das cartas jogadas durante toda a partida (incluindo valores de cartas de efeito aplicadas). Essa medida permite analisarmos a agressividade da IA durante a partida.

Outra medida é o número médio de movimentos por batalha, o qual mede a quantidade média de ações realizadas pela IA durante a partida, refletindo a eficiência no uso das jogadas pela IA.

A medida *mana* não utilizada representa a quantidade média de mana que restou sem uso ao final de cada turno. Esse indicador permite avaliar como cada IA gerencia seus recursos.

Por fim, o número de vitórias contabiliza quantas partidas cada IA venceu ao longo das simulações. Sendo um dos principais indicadores da eficácia da estratégia adotada, permitindo comparar objetivamente o desempenho das abordagens adotadas.

Para facilitar a análise dos resultados durante a seção, utilizaremos os acrônimos **IA. A** referindo-se a abordagem focada na maximização de poder e **IA. B** para a abordagem voltada à conservação de recursos. O código responsável pelo cálculo dessas medidas pode ser encontrado no **Apêndice A**.

Primeiramente, referente às médias de poder total acumulado durante as partidas, observou-se uma diferença significativa entre as duas abordagens. A **IA. A**, focada na maximização de poder, apresentou uma média de 25,39, significativamente superior à da **IA. B**, que foi de 22,77. Essa diferença reflete o foco da **IA. A** em utilizar cartas de maior impacto no dano causado ao oponente, buscando rapidamente reduzir os pontos de vida do adversário. Em contraste, a **IA. B**, com sua estratégia de conservação de recursos, prioriza um jogo mais equilibrado, o que resulta em um poder médio mais baixo.

Em relação ao número médio de movimentos por batalha, a **IA. A** realizou uma média de 8,78 movimentos por partida, enquanto a **IA. B** apresentou uma média ligeiramente maior de 9,2 movimentos. Esse dado indica que **IA. B** com seu foco em preservação de recursos, consegue manter um nível de *mana*, de forma a conseguir realizar mais movimentos consistentemente.

A análise da mana não utilizada também revela diferenças importantes entre as duas abordagens. A **IA. A** teve uma média de 0,296 unidades de mana não utilizadas durante as simulações, indicando sua eficiência no uso de recursos. Enquanto a **IA. B** apresentou uma média de 0,673 unidades de mana não utilizadas, indicando a abordagem de preservação de recursos para turnos futuros, permitindo uma flexibilidade na escolha das jogadas.

Por fim, em termos de vitórias, a **IA. A** destacou-se, conquistando um total de 60 vitórias em relação às 40 vitórias da **IA. B**. Esse resultado reforça a ideia de que a **IA. A**, ao focar na maximização do seu poder de combate durante os turnos, consegue causar dano mais rapidamente e, assim, vencer mais partidas.

Em resumo, os resultados mostram que **IA. A** tem uma vantagem em termos de poder total e velocidade de jogo, enquanto a **IA. B** apresenta uma abordagem mais equilibrada e estratégica, com um foco em partidas mais longas.

7. Dificuldades e Desafios

Durante o desenvolvimento deste trabalho, foram observados diversos desafios técnicos e dificuldades, exigindo ajustes e adaptações para garantir que o jogo fosse funcional ao fim do trabalho. Um dos principais desafios foi a definição das regras do jogo. Ao longo do processo, as mecânicas foram ajustadas diversas vezes para que o jogo não se tornasse excessivamente simples nem complexo demais.

Além disso, este projeto representou minha primeira experiência no desenvolvimento de jogos, o que trouxe dificuldades na implementação das mecânicas e na estruturação do código. Inicialmente, optei por utilizar a engine Godot como ferramenta de desenvolvimento. No entanto, devido à alta curva de aprendizado e ao tempo limitado para a conclusão do trabalho, decidi adotar uma abordagem mais simples e alinhada ao meu conhecimento, optando pelo desenvolvimento em Java.

Outro desafio, foi a criação dos recursos visuais e sonoros do jogo. Desenvolver os sprites foi uma experiência nova, exigindo aprendizado em alguns editores de imagem. Da mesma forma, integrar os efeitos sonoros e músicas foi uma experiência nova e proporcionou conhecimentos sobre manipulação de áudio com a linguagem Java.

Apesar dessas dificuldades apresentadas, o desenvolvimento deste trabalho proporcionou-me aprendizados valiosos na área de desenvolvimento de jogos e servirão como base para explorações futuras na área.

8. Conclusão

Este trabalho abordou o desenvolvimento de um jogo de cartas competitivo no estilo retrô, no qual foram implementadas e avaliadas duas abordagens distintas de inteligência artificial, a IA Focada na Maximização de Poder e a IA Focada na Conservação de Mana. Através de um conjunto de 100 partidas simuladas entre as IAs, foram analisadas as diferenças no desempenho das duas abordagens, onde os resultados mostraram que IA. A, focada na maximização de poder, apresentou um desempenho superior no número de vitórias e poder acumulado, enquanto a IA. B, com seu enfoque na gestão eficiente de mana, demonstrou uma abordagem mais equilibrada.

Para trabalhos futuros, sugerimos a exploração de algoritmos mais complexos, como o **MCTS**, que permitiria que as IAs tomassem decisões mais informadas, considerando uma quantidade maior de possíveis jogadas e cenários futuros. Outra abordagem, seria o aprendizado supervisionado, utilizando-se de dados de batalhas simuladas previamente. No que tange ao aprimoramento do jogo, é sugerido o aprofundamento das mecânicas existentes, como adição de novas cartas e efeitos especiais, e também melhorias visuais, com maior riqueza de animações e gráficos, tornando a experiência do usuário mais envolvente.

Em resumo, este trabalho forneceu uma base sólida para o desenvolvimento de jogos e permitiu uma avaliação detalhada das abordagens de IA aplicadas dentro desse contexto. A análise dos desafios enfrentados no desenvolvimento do jogo e sugestões para aprimoramentos futuros abrem caminhos para novas pesquisas e inovações, tanto na área de design de desenvolvimento de jogos quanto em inteligência artificial.

Referências Bibliográficas

Newzoo. (2023). **Global Games Market Report**. Disponível em: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version>. Acesso em: 22 dez. 2024.

Russell, Stuart J.; Norvig, Peter. **Inteligência Artificial - Uma Abordagem Moderna**. GEN LTC; 4ª edição (10 de agosto de 2022).

Yannakakis, Georgios N., and Julian Togelius. **Artificial Intelligence and Games**. Springer, 2018.

Schaeffer, Jonathan; Lake, Robert; LU, Paul; BRYANT, Martin. **Chinook: The World Man-Machine Checkers Champion**. *AI Magazine*, v. 17, n. 1, p. 21, 1996.

Campbell, Murray; Hoane Jr., A. Joseph; HSU, Feng-hsiung. **Deep Blue**. *Artificial Intelligence*, v. 134, n. 1-2, p. 57-83, 2002.

Silver, David et al. **Mastering the game of Go with deep neural networks and tree search**. *Nature*, v. 529, 2016. DOI: 10.1038/nature16961

SCHOLARWORKS SJSU. *Art 108: Game Design*. Disponível em: <https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1036&context=art108>. Acesso em: 22 dez. 2024.

TECHBULLION. *The Evolution of AI in Games: From Pixels to Deep Learning*. Disponível em: <https://techbullion.com/the-evolution-of-ai-in-games-from-pixels-to-deep-learning/>. Acesso em: 22 dez. 2024.

EPIC GAMES. *Behavior Tree in Unreal Engine - User Guide*. Disponível em: <https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-tree-in-unreal-engine---user-guide>. Acesso em: 22 dez. 2024.

V. GARCIA SC. *Simulador MCTS*. Disponível em: <https://vgarciasc.github.io/mcts-viz/>. Acesso em: 22 dez. 2024.

GITHUB. *V. Garcia SC - GitHub*. Disponível em: <https://github.com/vgarciasc>. Acesso em: 22 dez. 2024.

FREEPIK. *Pixel Art Rural Landscape Background*. Disponível em: https://www.freepik.com/free-vector/pixel-art-rural-landscape-background_49685499.htm#fromView=search&page=1&position=2&uuid=821b2e52-13f1-4bb5-b45f-f445c1f27ba9.

Acesso em: 22 dez. 2024.

FESLIYAN STUDIOS. *8-bit Music*. Disponível em: <https://www.fesliyanstudios.com/royalty-free-music/downloads-c/8-bit-music/6>. Acesso em: 22 dez. 2024.

SFXR. *Efeitos Sonoros*. Disponível em: <https://sfxr.me/>. Acesso em: 22 dez. 2024.

YouTube. (2024). [How to Make a 2D Game in Java]. Disponível em: https://www.youtube.com/watch?v=om59cwR7psI&list=PL_QPQmz5C6WUF-pOODsbsKbaBZqXj4qSq. Acesso em: 22 dez. 2024.

SHELL, J. The art of game design: a book of lenses. 3. ed. Boca Raton: CRC Press, 2019.

KOSTER, R. A theory of fun for game design. 2. ed. Sebastopol: O'Reilly Media, 2013.

JUUL, J. Half-real: video games between real rules and fictional worlds. Cambridge: MIT Press, 2005.

SUTTON, Richard S.; BARTO, Andrew G. Reinforcement Learning: An Introduction. 2. ed. Cambridge: MIT Press, 2018.

Apêndices

Apêndice A - Implementações

A.1 - Classe Estatística de Batalha

```
public class BattleStatistics {  
    private int totalTurns;  
    private int firstAiTotalPower;  
    private int secondAiTotalPower;  
    private int firstAiNuMovements;  
    private int secondAiNuMovements;  
    private int firstAiUnusedMana;  
    private int secondAiUnusedMana;  
    private String winner;  
}
```

A.2 - Simulação de Batalha entre IAs

```
public class Battle {  
    ...  
    public BattleStatistics startAiBattle() {  
        // AIs  
        AiEntity firstAi = (AiEntity) players.get(0);  
        AiEntity secondAi = (AiEntity) players.get(1);  
  
        // Medidas  
        int firstAiTotalPower = 0;  
        int secondAiTotalPower = 0;  
        int firstAiUnusedMana = 0;  
        int secondAiUnusedMana = 0;  
        int firstAiNuMovements = 0;  
        int secondAiNuMovements = 0;  
        int nuTurns = 0;  
  
        // Embaralhamos o baralho de ambos os jogadores e compramos 5 cartas
```

```

    for (PlayerEntity player : players) {
        player.getCardManager().shuffleDeck();
        player.getCardManager().drawCards(5);
    }

    while (BattleState.IN_PROGRESS == battleState) {
        // Turno da primeira AI
        handleTurnStart(firstAi);
        firstAiNuMovements += handleAiTurn(firstAi);
        firstAiUnusedMana += firstAi.getMana(); // Mana restante após o
turno

        firstAiTotalPower += firstAi.getFieldCard() != null ?
firstAi.getFieldCard().getTotalPower() : 0; // Função para calcular o dano
causado

        updateCurrentPlayerIndex();

        // Turno da segunda AI
        handleTurnStart(secondAi);
        secondAiNuMovements += handleAiTurn(secondAi);
        secondAiUnusedMana += secondAi.getMana(); // Mana restante após
o turno

        secondAiTotalPower += secondAi.getFieldCard() != null ?
secondAi.getFieldCard().getTotalPower() : 0;
        updateCurrentPlayerIndex();

        // Fase de combate
        handleBattlePhase();

        nuTurns++;

        // Atualizamos o estado da batalha
        updateBattleState();
    }

    String winner = firstAi.getHp() > 0 ? firstAi.getName() :
secondAi.getName();

```

```

        return new BattleStatistics(nuTurns, firstAiTotalPower,
secondAiTotalPower,
            firstAiNuMovements, secondAiNuMovements, firstAiUnusedMana,
secondAiUnusedMana, winner);
    }
}

```

A.3 - Service de Estatísticas de Batalhas

```

public class BattleStatisticsService {
    public static void runBattleSimulations(int numberOfBattles) {
        // Listas para armazenar as estatísticas de cada batalha
        List<BattleStatistics> battleStatsList = new ArrayList<>();

        for (int i = 0; i < numberOfBattles; i++) {
            Battle battle = new Battle(true);
            BattleStatistics stats = battle.startAiBattle();
            battleStatsList.add(stats);

            System.out.println("BATALHA " + String.valueOf(i + 1) + ": " +
stats);

            System.out.println(stats);
        }

        // Calculando as médias
        double totalTurns = 0;
        double totalFirstAiPower = 0;
        double totalSecondAiPower = 0;
        double totalFirstAiMovements = 0;
        double totalSecondAiMovements = 0;
        double totalFirstAiUnusedMana = 0;
        double totalSecondAiUnusedMana = 0;
        double totalFirstAiAverageMovementsPerTurn = 0;
        double totalSecondAiAverageMovementsPerTurn = 0;
        int firstAiWins = 0;
        int secondAiWins = 0;
    }
}

```

```

        for (BattleStatistics stats : battleStatsList) {
            totalTurns += stats.getTotalTurns();
            totalFirstAiPower += stats.getFirstAiTotalPower();
            totalSecondAiPower += stats.getSecondAiTotalPower();
            totalFirstAiMovements += stats.getFirstAiNuMovements();
            totalSecondAiMovements += stats.getSecondAiNuMovements();
            totalFirstAiUnusedMana += stats.getFirstAiAverageUnusedMana();
            totalSecondAiUnusedMana +=
stats.getSecondAiAverageUnusedMana();
            totalFirstAiAverageMovementsPerTurn +=
stats.getFirstAiAverageMovementsPerTurn();
            totalSecondAiAverageMovementsPerTurn +=
stats.getSecondAiAverageMovementsPerTurn();

            // Contando as vitórias de cada IA
            if (stats.getWinner().equals("IA Focada na Maximização de
Poder")) {
                firstAiWins++;
            } else {
                secondAiWins++;
            }
        }

        // Calculando as médias
        double averageTurns = totalTurns / numberOfBattles;
        double averageFirstAiPower = totalFirstAiPower / numberOfBattles;
        double averageSecondAiPower = totalSecondAiPower / numberOfBattles;
        double averageFirstAiMovements = totalFirstAiMovements /
numberOfBattles;
        double averageSecondAiMovements = totalSecondAiMovements /
numberOfBattles;
        double averageFirstAiUnusedMana = totalFirstAiUnusedMana /
numberOfBattles;
        double averageSecondAiUnusedMana = totalSecondAiUnusedMana /
numberOfBattles;

        // Exibindo os resultados médios
        System.out.println("===== Resultados Médios =====");

```

```
        System.out.println("- Turnos Médios: " + averageTurns);
        System.out.println("- Poder Médio da IA Focada na Maximização de
Poder: " + averageFirstAiPower);
        System.out.println("- Poder Médio da IA Focada na Conservação de
Mana: " + averageSecondAiPower);
        System.out.println("- Movimentos Médios da IA Focada na Maximização
de Poder: " + averageFirstAiMovements);
        System.out.println("- Movimentos Médios da IA Focada na Conservação
de Mana: " + averageSecondAiMovements);
        System.out.println("- Mana Média Não Utilizada da IA Focada na
Maximização de Poder: " + averageFirstAiUnusedMana);
        System.out.println("- Mana Média Não Utilizada da IA Focada na
Conservação de Mana: " + averageSecondAiUnusedMana);
        System.out.println("- Vitórias da IA Focada na Maximização de Poder:
" + firstAiWins);
        System.out.println("- Vitórias da IA Focada na Conservação de Mana: "
+ secondAiWins);
    }
}
```