Universidade Federal do ABC
Centro de Matemática, Computação e Cognição (CMCC)
Pós-Graduação em Ciência da Computação

Jonathan Ohara de Araujo

AGENTES INTELIGENTES PARA BATALHAS POKÉMON

Dissertação

Santo André - SP
2016

Jonathan Ohara de Araujo

AGENTES INTELIGENTES PARA BATALHAS POKÉMON

Qualificação

Qualificação apresentada ao Curso de Pós-Graduação da Universidade Federal do ABC como requisito parcial para obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Fabrício Olivetti de França

Santo André - SP

2016

Ficha Catalográfica

de Araujo, Jonathan Ohara.
Agentes inteligentes para batalhas Pokémon / Jonathan Ohara de Araujo.
Santo André, SP: UFABC, 2016.
3p.

Jonathan Ohara de Araujo

AGENTES INTELIGENTES PARA BATALHAS POKÉMON

Essa Qualificação foi julgada e aprovada para a obtenção do grau de Mestre em Ciência da Computação no curso de Pós-Graduação em Ciência da Computação da Universidade Federal do ABC.

Santo André - SP - 2016

Prof. Dr. João Paulo Góis

Coordenador do Curso

BANCA EXAMINADORA

Prof. Dr. Fabrício Olivetti de França	Prof.Dr. André Luiz Brandão
Prof.Dr. João Paulo Gois	Prof.Dr. Denise Hideko Goya (Suplente)

Resumo

Esse trabalho explora a criação de agentes inteligentes competitivos em um ambiente onde dezenas de milhares de jogadores humanos competem diariamente para melhorar sua colocação no sistema de ranqueamento do jogo Pokémon Showdown.

Escolher o melhor algoritmo e a melhor forma de aprendizado para jogar contra humanos é um dos grandes desafios desse trabalho. Outro importante aspecto que o agente precisa se adaptar é a grande variedade de composições de times e Pokémons que o agente e o seu adversário pode montar.

Para obter a melhor flexibilidade o agente será submetido somente a batalhas randômicas onde as composição das equipes são todas aleatórias, e a avaliação de performance do agente será feita através do sistema de ranqueamento do próprio jogo.

Abstract

This work explores the creation of competitive intelligent agents in an environment where thousands of human players compete every day to improve their placement in the ranking system of the Pokémon Showdown game.

Choosing the best algorithm and the best learning method against human is one of the great challenges of this work. Another important aspect that the agent needs to adapt is the wide variety of teams compositions and Pokémon customization that the agent and your opponent can build.

To get more flexibility the agent will be subject only to random battles where team compositions are all random, and the agent performance evaluation will be done through the game ranking system.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o aumento do poder computacional e a viabilidade de grandes empresas em construírem clusters de máquinas para processamento paralelo fomentou grandes avanços na área de Inteligência Artificial (IA), permitindo construções de algoritmos mais complexos e obtenção de resultados em larga-escala.

Uma das subáreas de estudo de interesse em IA é a construção de agentes inteligentes capazes de tomar decisões de maneira autônoma para as mais diversas tarefas. Como exemplo é citado tarefas como geração de conteúdo procedural para jogos [Hendrikx et al., 2013], geração de missões e desafios [Doran and Parberry, 2011], agentes para jogos eletrônicos [van Lent et al., 1999], dentre outras.

Especificamente sobre os agentes criados para jogos eletrônicos, temos que a tarefa desses agentes é a de vencer desafios lógicos ou de precisão de tal forma a vencer um determinado jogo eletrônico, o que geralmente é desafiador dadas as possíveis entradas e estados de saída, além de múltiplas formas de vencer o desafio.

A indústria de jogos eletrônicos, segundo [GameIndustry, 2016], teve uma receita de cerca de 90 bilhões de dólares em 2015 e de acordo com projeções do site [PWC, 2016] crescerá dentre 5 e 6% por ano até 2019. Um dos principais interesses dessa indústria é a de criar jogos que capturem a atenção do jogador pelo maior período de tempo possível, para isso os jogos devem apresentar o nível adequado de desafio. Segundo [Millington and Funge, 2009] existe um grande número de jogos em que o diferencial é sua inteligência artificial.

Nesses pontos, um agente inteligente pode contribuir sendo treinado para evoluir seu desafio juntamente com a evolução do jogador ou para testar a viabilidade em se completar um determinado estágio do jogo.

Além da indústria de jogos, o estudo de agentes inteligentes em jogos eletrônicos é de interesse da área acadêmica, pois é possível criar desafios adaptados para diversos interes-

ses e de diversos níveis de dificuldade. Por conta disso, surgiu uma comunidade científica para estudar a construção de agentes inteligentes em jogos eletrônicos (Computational Intelligence in Games - CIG) e foram criadas diversas competições onde técnicas poderiam ser testadas e comparadas.

Um problema que essas competições compartilham é a impossibilidade de treinar esses agentes em um ambiente que contenham muitos jogadores humanos, dificultando assim desenvolver uma IA competitiva contra bons jogadores.

Para resolver essa questão, esse trabalho propõe a criação de uma API de comunicação com o jogo *Pokémon Showdown!*, um simulador *on-line* de batalhas *Pokémon*. *Pokémon Showdown!* é um ambiente totalmente *on-line* onde joga-se apenas contra jogadores reais, existem vários modos de jogo, todos eles são um contra um jogador, cada usuário tem uma pontuação nos diferentes modos de jogo e o sistema encontra adversários com pontuação semelhante para realizar as batalhas.

1.1 Motivação

Existem diversas finalidades para criação de agentes inteligentes para jogos. Podemos enumerar algumas como: testar se é possível um agente jogar um jogo tão bem quanto um jogador humano, testar algoritmos de aprendizados para agentes, criar agentes que se comportem como humanos entre outras finalidades.

Esse trabalho explora a criação de agentes inteligentes que compitam em alto nível com jogadores humanos no sistema de batalhas *Pokémon*.

Um dos grandes desafios na criação desses agentes é a adaptabilidade e a competitividade. Diferentes de jogos como xadrez, damas, *reversi* e outros jogos de tabuleiro, em batalhas *Pokémon* pouco se sabe do adversário até que comece o jogo, ou seja, invalidando qualquer tipo de técnica prevendo possíveis movimentos antes do jogo começar, já que cada time pode ser composto de uma infinidade de maneiras diferentes (no capítulo 4 é apresentado com detalhes essas possiblidades). Para acentuar ainda mais essas propriedades os agentes irão treinar e competir no modo randômico, nesse modo a escolha do seu time e de seu adversário é feito pelo próprio sistema do jogo.

Por causa da característica de desconhecimento do time adversário, a utilização de técnicas de criação de árvores de possíveis jogadas do adversário é bastante prejudicada, pois o agente precisaria predizer os possíveis *Pokémons* adversários assim como suas características, dificultando a utilização da técnica que ficou famosa pelo sistema *Deep Blue* que segundo o trabalho *Deep Blue System Overview* [Hsu et al., 1995] "O *Deep Blue* é um massivo sistema paralelo para realização de busca em árvores de jogos de xadrez".

1.2 Objetivos

O Objetivo desse trabalho é o desenvolvimento de agentes inteligentes que joguem e aprendam com milhares de jogadores humanos. Serão implementadas distintas técnicas para criação e aprendizado desses agentes. Durante o desenvolvimento da pesquisa será sumarizado a evolução dos agentes no sistema de ranqueamento do jogo e, essa posição será confrontada com a quantidade de treinamento que cada agente recebeu, podendo assim observar a curva de melhora em relação a quantidade de treinamento.

Para criação desses agentes foi desenvolvida uma API que permitirá a comunicação com o jogo *Pokémon Showdown*. Inicialmente a API está disponibilizada apenas para linguagem JavaScript, mas durante o desenvolvimento do projeto será portada para Java através da tecnologia de WebSockets.

1.3 Principais contribuições

O trabalho irá explorar a criação de agentes inteligentes adaptativos, num ambiente que pouco se sabe sobre o adversário. Os agentes terão informações apenas durante a batalha e, essas informações são apenas aquelas que o adversário realizar, por exemplo, o agente só saberá que adversário tem um *Pokémon* até o mesmo usá-lo, os movimentos que o *Pokémon* tem serão apenas conhecidos a medida que o adversário utilizá-los e mesmo assim existirá características que agente não tem como descobrir, como por exemplo a quantidade de ataque e defesa distribuída no monstro.

Além disso, a construção de uma API para acesso ao jogo contribuirá para que outros pesquisadores também possam desenvolver estudos e criar seus próprios agentes podendo criar-se uma cultura de competição entre agente inteligentes na plataforma *Pokémon Showdown*.

Capítulo 2

Agentes Inteligentes

Na computação, especialmente em inteligência artificial, é bem comum utilizar-se do termo agente ou agente inteligente. Por definição do dicionário [Michaelis, 2016] agente significa "Que age, que exerce alguma ação; que produz algum efeito".

Na academia temos definições mais direcionadas como a de [Russell and Norvig, 2010] "Um agente é algo capaz de perceber seu ambiente através de sensores e agir sobre esse ambiente por meio de atuadores", ou a de [Smith et al., 1994] "Vamos definir um agente como uma entidade de *software* persistente dedicada para um propósito específico. 'Persistente' distingue agentes de sub-rotinas; agentes têm seus próprias ideias sobre como realizar tarefas, suas próprias agendas. 'Propósitos especiais' os distingue de aplicações multifuncionais inteiras; agentes são tipicamente muito menores."

Existem diversas definições de agentes. Mas qual seria a diferença de um agente para um programa de computador? Os autores [Franklin and Graesser, 1997] propõem uma definição mais sólida para agente "Um agente autônomo é um sistema situado dentro e como parte de um ambiente que sente esse ambiente e age sobre ele e, ao longo do tempo, busca de sua própria agenda de modo a efetivar o que sente no futuro."ainda segundo os autores maioria dos programas comuns violam uma ou mais regras dessa definição, por exemplo um sistema de folha de pagamento sente o ambiente através de suas entradas e age sobre ele gerando uma saída, mas não é um agente porque sua saída normalmente não afeta o que ele sente mais tarde.

2.1 Classificações de agentes

Os agentes possuem também diferentes explicações quanto a suas características, classificações e taxonomia. Uma das definições mais antigas e mais sólidas é a de [Nwana, 1996] nesse trabalho o autor define três dimensões para o agente:

- **Mobilidade**. Capacidade de se mover em torno de alguma rede. Definindo agentes como fixos ou móveis;

- **Deliberativos ou reativos**. Agentes deliberativos possuem símbolos internos, modelos de raciocínio e se envolvem com o planejamento e negociação a fim de conseguir uma coordenação com outros agentes. Já os reativos agem por estímulos/respostas que o estado atual do ambiente que está inserido o proporciona.

- **Ideais e atributos primários**. Os agentes podem ser classificados de acordo com três características: autonomia, cooperatividade e aprendizagem. Autonomia se refere ao princípio que o agente deve realizar ações sem intervenção humana. Cooperação é a habilidade de se comunicar com outros agentes. Aprendizagem é a capacidade de aprender conforme informações são lidas e ações sejam realizadas. Dessas definições surgem quatro tipos de agentes conforme mostra a Figura 2.1.

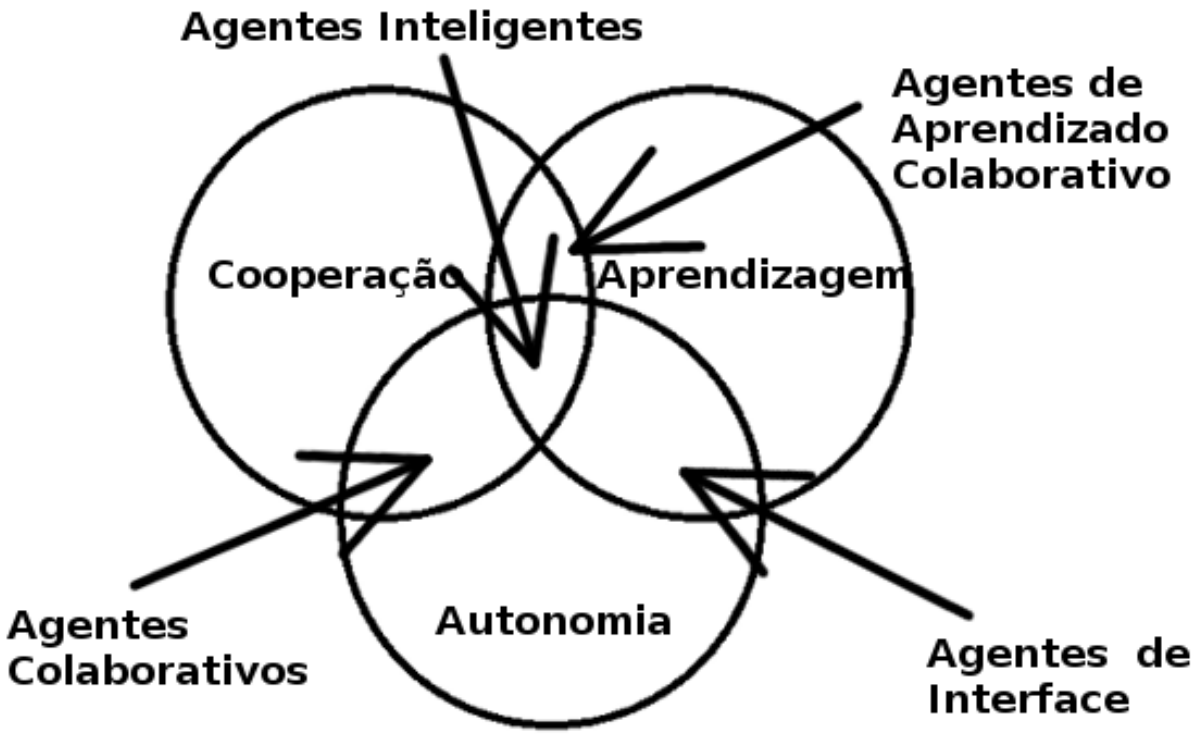

Figura 2.1: Visão parcial da tipologia de um Agente.

Já [Franklin and Graesser, 1997] classifica os agentes pelas propriedades que eles apresentam, são elas: reatividade, autonomia, orientação a objetivo, temporária ou contínua, comunicativa, aprendizagem, mobilidade, flexibilidade e caráter.

Embora as diferentes definições, muitas coisas são comuns ou muito parecidas como a presença de características como autonomia, aprendizado, objetivos e comunicabilidade.

2.2 Agentes em jogos

Antigamente maioria do processamento do sistema que rodava um jogo (computador, *videogame* entre outros) era utilizada para processamentos gráficos. Com a evolução e custo baixo dos equipamentos (*hardware*) hoje em dia é possível dar mais atenção a outros aspectos de um jogo. O autor [van Lent et al., 1999] discorre um outro ponto "Como jogos de computador se tornam mais complexos e os consumidores exigem adversários mais sofisticados controlados pelo computador, os desenvolvedores de jogos são obrigados a colocar uma maior ênfase nos aspectos de inteligência artificial de seus jogos". Por isso hoje, existem um grande número de jogos que não são conhecidos por sua beleza gráfica, segundo [Millington and Funge, 2009] um número cada vez maior de jogos tem como principal ponto do jogo a Inteligência Artificial, como por exemplo Creatures[Cyberlife Technology Ltd., 1997] em 1997, e jogos como The Sims[Maxis Software, Inc., 2000] e Black and White [Lionhead Studios Ltd., 2001]. Ainda segundo [Millington and Funge, 2009] o jogo Creatures foi um dos sistemas mais complexos de inteligência artificial visto em um jogo, com um cérebro baseado em rede neural para cada criatura presente no jogo.

Agentes para jogos tem uma infinidade de propósitos. Ao associar esses dois termos é bem comum pensar em oponentes controlados por agentes, porém a utilização de agentes vai muito além. É possível utilização de agentes em outros aspectos como na geração de conteúdo procedural, criação de desafios e missões entre outros.

O artigo *Procedural Content Generation for Games: A survey* [Hendrikx et al., 2013] cita que a geração de conteúdo através de agentes inteligentes permite a criação de diferentes tipos de terrenos sem perder o controle do *design* do jogo.

O trabalho de [Doran and Parberry, 2011] explora a geração procedural de missões e desafio em jogos(especialmente para jogos de MMORPGs *online*), segundo os autores essa abordagem tem o potencial de aumentar a variedade e a longevidade de um jogo.

Existem também outras aplicações para agentes como busca de caminhos em ambientes complexos, geração de objetos, danificação de objetos entre outras.

2.3 Agentes inteligentes contra jogadores

Existem várias abordagens para criação de agentes que enfrentam jogadores. Podem-se enumerar alguns deles como agentes adaptativos, evolutivos e competitivos.

Em alguns jogos o jogador não tem uma opção de dificuldade, ficando a critério do sistema do jogo controlar a dificuldade, como nos jogos *Middle Earth: Shadow of Mordor* e *Max Payne 2*. Segundo [Ponsen and Spronck, 2004] no jogo Max Payne 2 é introdu-

zido algo chamado opções dinâmicas de dificuldade. Informações do mundo do jogo são extraídas para estimar a habilidade do jogador, ajustando a dificuldade da inteligência artificial.

Alguns agentes são criados para acompanhar a curva de aprendizado de cada pessoa, para que o jogador sempre tenha um bom nível de desafio sempre que jogar. No jogo *Star Craft 2* existe o modo chamado pareamento contra Inteligência Artificial, nesse modo o desempenho do jogador em partidas passadas define qual o nível do jogador, ou seja, quanto mais o jogador evolui mais difícil será seu oponente.

Outra abordagem interessante é a utilizada em Drivatar [Herbrich, 2011], inserida no jogo Forza Motorsport, é um modulo que cria agentes "humanizados"que aprendem com a pessoa que está jogando, incorporando suas características e utilizando para jogar contra o próprio jogador.

Este trabalho explora a criação de agentes competitivos, ou seja, agentes projetados para ganhar de jogadores humanos. Segundo [Schaeffer and van den Herik, 2002] apenas na década de 90 os computadores foram capazes de competir com sucesso contra o melhor dos humanos. O primeiro jogo a ter um campeão mundial não humano foi o jogo de damas em 1994 seguido pelo Deep Blue em 1997.

O agente jogador mais conhecido é o Deep Blue da IBM. O artigo Deep Blue: System overview [Hsu et al., 1995] explica o funcionamento do agente. O Deep Blue é composto por um *chip* chamado *Chess Chip* que é dividido em três partes:

- **Buscador alphabeta**. Menor parte do *chip*, contendo apenas 5% de seu total, esse componente é responsável por fazer a busca na árvore de jogadas. O algoritmo utilizado é uma variante do *alphabeta* chamado de *minimum-window alpha beta search*, essa técnica foi escolhida por não ser necessário uma pilha de valores.

- **Gerador de movimentos**. Esse componente contém o *array* bidimensional 8x8 referente a cada posição do tabuleiro do xadrez, ele também é responsável por realizar e verificar a legalidade dos movimentos de acordo com as regras do xadrez.

- **Função de avaliação**. Ocupando cerca de dois terços do *chip* esse componente é responsável pela avaliação dos movimentos. Cada possível posição de cada peça é avaliado, essa avaliação é baseada em quatro critérios: material, posição, segurança do rei e tempo [Works, 2016]. Material é o quanto cada peça "vale". Por exemplo, um peão vale 1 enquanto uma torre vale 5. Posição quantifica o quão bom estão posicionadas as peças, nesse cálculo uma série de aspectos é levada em conta, um deles é o número de movimentos seguros que as peças podem fazer para atacar. Segurança do rei é calculo que leva em conta a proteção do rei. Por final tempo, além

de controlar o tempo da própria jogada, fazer uma jogada rápida dá ao adversário menos tempo para pensar em possíveis jogadas.

Como pode ser visto no agente do Deep Blue a heurística é algo muito importante na construção de um agente para jogos complexos. Em alguns jogos a avaliação de heurística não é tão importante, um exemplo desse tipo de jogo é o jogo da velha, nele temos uma árvore pequena de possibilidades e o resultado é vitória, empate ou derrota, ou seja, não é necessário que a cada nó da árvore seja feita uma complexa avaliação de heurística.

Segundo [Nielsen, 1994] avaliação heurística envolve ter um pequeno conjunto de avaliadores examinando a interface e avaliando a sua conformidade em relação ao resultado.

Os autores [Christensen and Korf, 1986] discorrem que, em jogos de tabuleiro de duas pessoas a função de heurística é vagamente caracterizado pela ”força”do posicionamento de um jogador contra o outro. Ainda segundo [Christensen and Korf, 1986] pode-se dizer que uma função de avaliação de heurística tem duas propriedades:

- Quando aplicado a um estado final (no caso de uma árvore, um nó folha), a avaliação de heurística tem que devolver o estado corretamente;
- O valor da função de heurística é invariável ao longo de um caminho de solução ótima.

2.4 Competição de agentes inteligentes

Um recurso bastante utilizado para pesquisa em inteligência artificial é o desenvolvimento de agentes inteligentes que compitam com outros agentes que utilizem diferentes técnicas. Tais competições são bem comuns nos grandes congressos de inteligência artificial.

As competições de inteligência para Start Craft que ocorrem desde 2010 avançou significativamente o campo de inteligência artificial para jogos de estratégia em tempo real segundo [Ontanon et al., 2013]. Em Star Craft os jogadores tem que se preocupar em construir um exército ao mesmo tempo em que gere uma economia bem complexa. Nesse jogo o vencedor é definido pelo jogador que derrotar todas as tropas e estruturas adversárias. Nessa competição dois agentes são colocados para guerrear até que haja um vencedor. A comunicação com o jogo é feito pela BWAPI uma API em C++ que permite comunicação do agente com o jogo, ou seja, através dessa biblioteca é possível realizar comandos que serão executados no jogo.

A figura 2.2 mostra a API funcionando. Ele usa o sistema de conversa do jogo para mostrar mensagens do agente.

Figura 2.2: BWAPI em funcionamento.

Outra competição um pouco diferente é a AI Birds [AIBirds, 2016], nessa competição os agentes são submetidos a jogar Angry Birds, o objetivo desse jogo é você destruir objetos(ganhando assim pontos) com pássaros que são arremessados por um estilingue. Diferente do Star Craft nessa competição não há um enfretamento direto entre os agentes, eles são submetidos a enfrentar o mesmo cenário e vence o agente que obtiver mais pontos. A comunicação com o jogo é bastante diferente do Star Craft, nesse caso o jogo roda via navegador de internet (apenas Google Chrome) e um *plugin* de JavaScript tira fotos do jogo e passa essa imagem via WebSocket para uma linguagem que irá interpretar tal foto.

Na figura 2.3 pode ser visto como a API se comunica com jogo. Uma imagem é capturada do navegador e a partir disso é utilizado um algoritmo de reconhecimento de imagem para identificar o que é cada objeto (os quadrados em volta do objeto são padrões que a API está reconhecendo).

Neste trabalho além da criação de agentes, será explorada a criação de uma API para comunicação com o jogo Pokémon Showdown, um simulador de batalhas Pokémon. No capítulo 4 será explicado como funciona o sistema de batalhas. A API possibilita batalhas

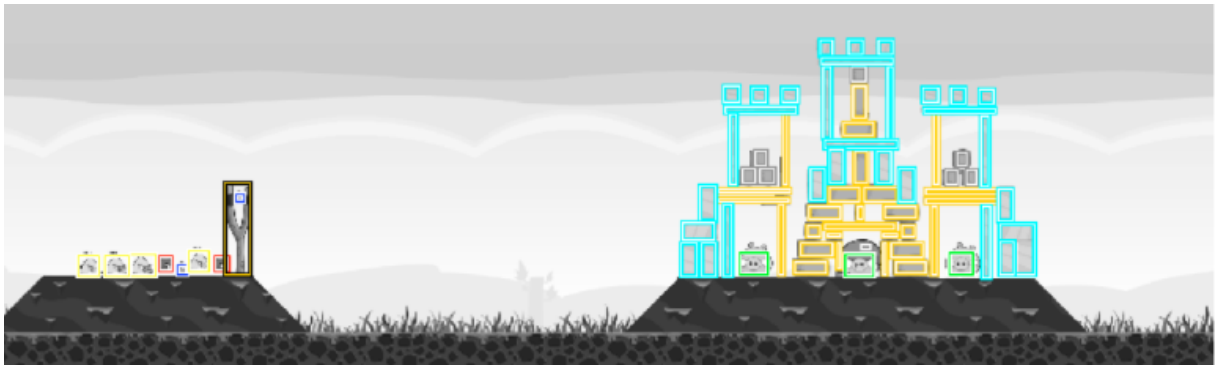

Figura 2.3: Api Angry Birds AI em funcionamento.

contra outros agentes e contra jogadores humanos. A API está escrita em JavaScript, mas também haverá possiblidade de comunicação via WebSocket, onde qualquer linguagem que tenha o recurso poderá fazer um agente.

Capítulo 3

Inteligência Artificial

Neste capítulo é feito uma revisão das técnicas de inteligência artificial que serão utilizadas nesse trabalho. Algoritmos baseados em grafos 3.2, aprendizado por reforço 3.3 e neuroevolução 3.4.

3.1 Inteligência para jogos

O estudo de IA/IC (Inteligência artificial/Inteligência computacional) para jogos é uma área de estudo que vem crescendo muito na última década. Grandes conferências como o IEEE *Conference on Computational Intelligence and Games* (CIG), AAAI *Aritificial Intelligence and Interactive Digital Entertainment* (AIIDE) e IEEE *Transactions on Computational Intelligence and AI in Games* (TCAIG) já contam com mais de dez edições.

Jogos podem ser usados como cenário desafiador para avaliação de métodos de inteligência computacional, pois eles provêm elementos dinâmicos e competitivos que são pertinentes ao mundo real [CIG, 2016].

Segundo [Yannakakis and Togelius, 2015] durante os seminários em Dagstuhl (centro de pesquisa de ciência da computação na Alemanha) foi possível identificar dez grandes tópicos em IA/AC:

- Aprendizado de comportamento para jogadores não humanos (JNH);
- Busca e planejamento;
- Modelagem de personagens;
- Jogos como avaliação comparativa para Inteligência Artificial (IA);
- Geração de conteúdo procedural;

- Narrativa computacional;
- Agentes críveis;
- *Game design* assistido por IA;
- IA para jogos em geral;
- IA em jogos comerciais.

Ainda segundo [Yannakakis and Togelius, 2015] todas as áreas de pesquisa podem ser vistas como potenciais influenciadores delas mesmas em algum grau. Essas conexões e interconexões geram outras áreas de pesquisa.

Nas próximas seções será feito uma revisão dos principais métodos que serão utilizados nos agentes deste trabalho.

3.2 Algoritmos baseados em grafos

Um dos grandes desafios de um agente e de um jogador dentro de um jogo é a tomada de decisão, expandindo um pouco essa ideia, pode-se dividir a tomada de decisão em partes menores: levantar opções, avaliar as opções e escolher qual pode conduzir o jogador a vitória dado um determinado cenário.

Uma algoritmo muito comum para esse cenário é o *minimax*. Segundo os autores [aend Edward Powley et al., 2012] jogos do mundo real normalmente envolvem uma estrutura de recompensas em que apenas as recompensas obtidas em estados terminais (jogadas que definem quem é o vencedor) do jogo descrevem com precisão o quão bem cada jogador está se saindo. Os jogos são, portanto, normalmente modelado como as seguintes árvores de decisões:

- **Minimax** tenta minimizar recompensa máxima do oponente em cada estado, e é a abordagem tradicional para pesquisa de jogos combinatórios em dois jogadores.
- **Expectimax** generaliza *minimax* para jogos estocásticos em que as transições de estado para estado são probabilística. O valor de um nó é a soma dos valores dos nós filhos ponderados por suas probabilidades (possibilidade de um estado ocorrer). Estratégias de poda de árvore são mais complexas devido à probabilidade de um nó acontecer.
- **Miximax** é semelhante ao *expectimax* de apenas um jogador e é usado principalmente em jogos com informações não precisas. Ele utiliza uma estratégia predefinida para tratar a decisão do oponente como nós probabilísticos.

O trabalho de [Campbell and Marsland, 1983] descreve o algoritmo de *minimax* do seguinte modo: O algoritmo assume que existem dois jogadores chamados Max e Min, e atribui um valor para cada nó dentro da árvore (de decisão) do jogo (e também para nó raiz) do seguinte modo: Nós folhas ou terminais podem propagar o valor *minimax* recursivamente. Se a jogada p do jogador Max for escolhida, então o valor de p é o máximo valor dos filhos de p. Similarmente, se for o turno do jogador Min será escolhido o menor valor dos sucessores de p.

No trecho abaixo é apresentado o algoritmo do *minimax*. O algoritmo analisa todas as possíveis jogadas até que sejam encontrados todos os nós que finalizam o jogo e retorna um valor chamado de heurística representando vitória, derrota ou empate.

Algorithm 1 Algoritmo Minimax

```
1:  procedure MINIMAX(node, depth, max)
2:      if depth = 0 OR node is terminal then
3:          return heuristic
4:      end if
5:      if max then
6:          bestValue ← −∞
7:          for each child ∈ node do
8:              value ← MINIMAX(child, depth − 1, FALSE)
9:              bestValue ← MAX(bestValue, value)
10:         end for
11:     else
12:         bestValue ← +∞
13:         for each child ∈ node do
14:             value ← MINIMAX(child, depth − 1, TRUE)
15:             bestValue ← MIN(bestValue, value)
16:         end for
17:     end if
18:     return bestValue
19: end procedure
```

Onde:

- **node** Nó atual da árvore. Na primeira chamada é passado o nó raiz da árvore, ou seja, o estado atual do jogo.
- **depth** Profundidade do nó. Inicialmente é passada a altura da árvore.
- **max** Booleano representando se a jogada analisada é do jogador **Max** ou **Min**.

Na figura 3.1 é mostrado a aplicação do algoritmo de *minimax* para um cenário do jogo da velha. Na imagem o jogador Max é representado pelo caractere **X** e o jogador Min por **O**. Como dito anteriormente, a análise dessa árvore começou a ser feita pelos nós

terminais, em situações vitoriosas foi atribuído o valor de 10, em empates 0 e em derrota o valor -10. Em seguida, os pais desses nós folhas são preenchidos com o mínimo ou o máximo valor dos nós filhos. No primeiro nó de MIN (primeiro nó da segunda fileira), por exemplo, existem dois nós filhos, um com valor 10 e outro com valor -10, por ser um nó MIN é atribuído para esse nó o menor valor (no caso -10). A linha em azul mostra a melhor jogada no momento para o nó raiz.

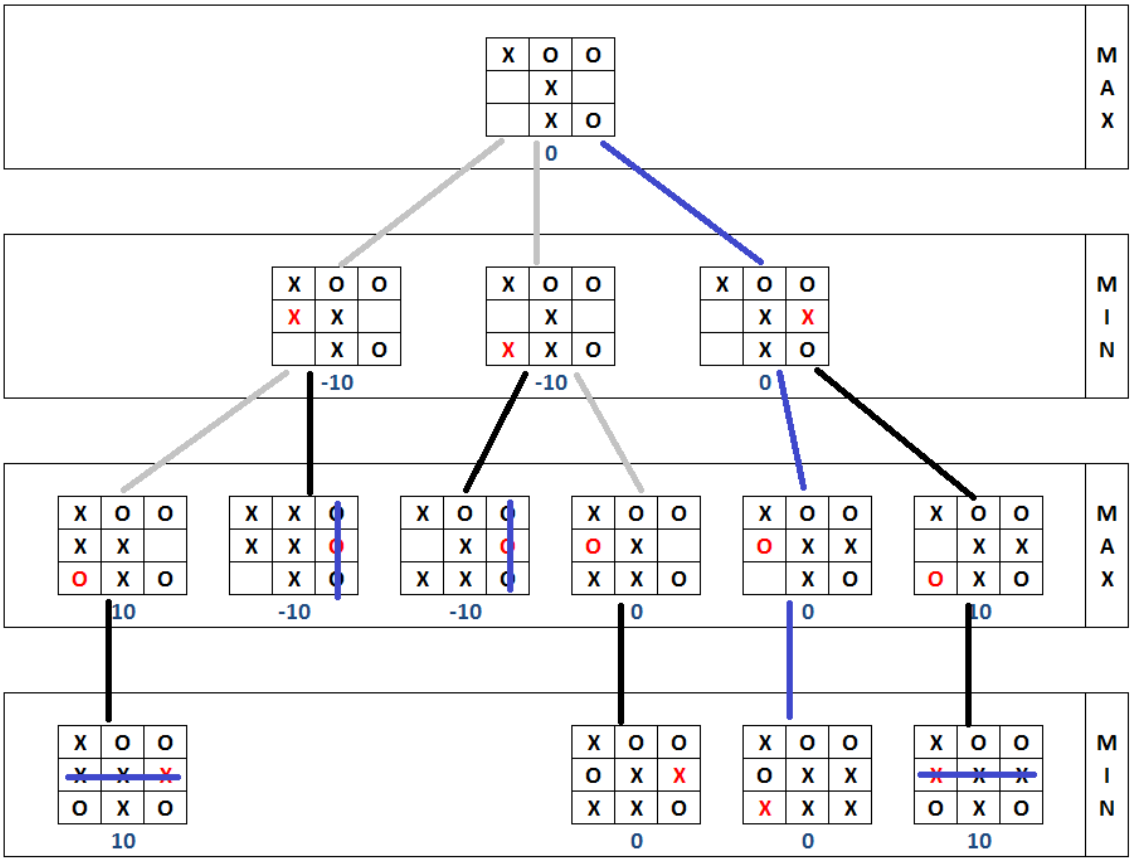

Figura 3.1: *Minimax* aplicado a um cenário de jogo da velha.

Repare que a melhor jogada indicada pelo algoritmo gera um empate. Segundo [Moriarty and Miikkulainen, 1994] um dos problemas do *minimax* é presumir que sempre o adversário irá fazer a melhor escolha possível. Muitas vezes, em situações de derrota iminente a melhor jogada pode não ser o valor mais alto de *minimax*, especialmente se ele ainda irá resultar em uma derrota.

Outro problema do *minimax* é a quantidade de nós que ele precisa analisar, em jogos com grande número de possíveis jogadas ou que duram um grande número de turnos (grande profundidade na árvore) a quantidade de nós que ele precisa analisar é muito grande.

Uma solução comum para esse problema é definir um número máximo de profundidade

para analisar, ou seja, ao invés do critério de parada do algoritmo recursivo ser nós folhas (terminais) pode-se definir uma profundidade máxima para o *minimax*. Nessa abordagem muitas vezes não serão encontrados nós que representam fim de jogo, então é necessário fazer uma função de avaliação de heurística que analisa o estado do jogo, como foi apresentado na seção 2.3.

Outra técnica muito utilizada para mitigar esse problema é a chamada poda *alpha-beta*. Segundo [Knuth and Moore, 1976] essa é técnica é usada geralmente para aumentar a velocidade de busca sem perder informação. Nessa técnica são ignorados os nós e suas subárvores de jogadas incapazes de ser melhor do que movimentos já conhecidos. Durante a análise das jogadas são definidas duas variáveis *alpha* e *beta*. *Alpha* representa o valor máximo que o jogador Max pode fazer e *beta* a pontuação mínima do jogador Min. A cada avaliação de nó esses limites são verificados, caso o valor da avaliação não estiver entre esses limites o nó e todas suas árvores são cortados.

Outra técnica de otimização do *minimax* é a Árvore de Busca de Monte Carlo(MCTS). Segundo [aend Edward Powley et al., 2012] o processo de construção da MCTS é feita de modo incremental e assimétrico na seguinte forma: para cada iteração do algoritmo, uma ”política de árvore”é utilizada para definir o nó mais urgente da árvore atual. A política da árvore tenta balancear considerações da exploração (procura áreas que ainda não tenham sido bem amostradas) e aprofundamento (procura áreas que parecem ser promissoras). O nó escolhido é avaliado e todos seus ancestrais são atualizados com suas novas estatísticas.

Uma das grandes vantagens do MCTS é a possibilidade de utilizar de maneira incremental, ou seja, é possível delimitar um tempo ou número máximo de iterações que o algoritmo irá rodar, facilitando a aplicação em ambientes que exigem baixo tempo de resposta (jogos de tempo real) ou tempo de resposta fixado (jogos baseados em turno).

3.3 Aprendizado por reforço

Uma técnica bastante utilizada em aprendizado de máquina é o aprendizado por reforço (RL). Segundo [Sutton and Barto, 1998] a ideia de aprender interagindo com nosso ambiente provavelmente é a primeira coisa que nos ocorre quando pensamos sobre aprendizado natural.

Segundo [Kaelbling et al., 1996] o algoritmo de aprendizado por reforço é um modo de programar agentes por um sistema de punição e recompensa sem precisar especificar como a tarefa precisa ser realizada.

Segundo [Mitchell, 1997] o aprendizado por reforço pode resolver tarefas como aprender a controlar um robô móvel, aprender a otimizar operações em fábricas, e aprender a

jogar jogos de tabuleiros.

Novamente segundo [Kaelbling et al., 1996] existem duas principais estratégias para resolver problemas com aprendizado por reforço. A primeira é uma busca entre os possíveis comportamentos para achar aquele que se adequa melhor ao ambiente. O Segundo modo é utilizar métodos estocásticos e métodos de programação dinâmica para estimar a utilidade de tomar ações em estados do mundo.

A figura a 3.2 mostra o funcionamento básico do algoritmo. O agente está em um determinado estado e executa uma ação recebendo uma recompensa ou punição e vai para um novo estado.

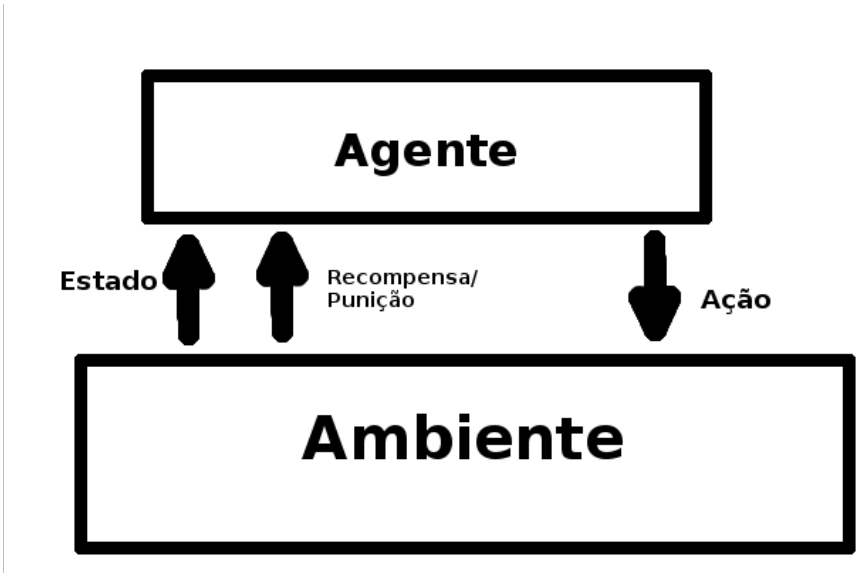

Figura 3.2: Esquema padrão do aprendizado por reforço.

Existe uma série de alterações nessa abordagem para abranger uma maior gama de problemas. Segundo [Mitchell, 1997] existe alguns aspectos diferentes ao considerar utilizar aprendizado por reforço para alguns problemas:

- **Recompensa atrasada**. Grandes recompensas podem estar somente em estados que serão apenas alcançados num futuro longínquo.
- **Exploração**. No aprendizado por reforço, o agente influencia a distribuição do treino pela sequência de ações que escolhe. Com isso surge a dúvida, qual das estratégias produz o mais efetivo aprendizado (para colher novas informações). Explorar todas as possíveis ações, ou explorar estados e ações já conhecidos com alta recompensa (para maximizar a sua recompensa cumulativa)?
- **Estados parcialmente observáveis**. Em muitos casos os sensores do agente não consegue observar o estado inteiro do ambiente. Por exemplo, um robô com câmera frontal não pode enxergar o ambiente que está atrás dele.

- **Aprendizagem ao longo da vida**. Possibilidade de utilizar conhecimentos obtidos anteriormente para reduzir a complexidade de aprender novas Tarefas.

Uma das técnicas que resolve alguns desses problemas é a chamada processo de decisão de Markov(MDP). Segundo [Kaelbling et al., 1996] consiste em:

- Um conjunto de estados chamado S,
- Um conjunto de ações chamado A,
- Uma função de recompensa onde $R : S \times A \rightarrow \mathbb{R}$,
- Uma função de transição entre os estados onde $T : S \times A \rightarrow II(S)$.

A função de transição de estados específica o próximo estado como uma função do estado atual e a ação do agente. Para que isso seja válido o ambiente tem que satisfazer a propriedade de Markov, que diz que a transição de estados tem que ser independente de qualquer informação de estados e ações dos agentes anteriores.

Com essas propriedades é possível calcular qualquer estado futuro pela função de transição de estados resolvendo o problema de recompensa atrasada, como é possível calcular todos os possíveis futuros e estados e ações também é possível calcular futuras recompensas.

Uma técnica muito utilizada para melhorar o processo de escolher qual política de ações é o Q-Learning. Segundo [Kaelbling et al., 1996] no Q-Learning os valores de Q (valor de escolher uma ação a em um estado s) irá convergir para valores ideais, independente de como o agente se comporta enquanto os dados estão sendo coletados (desde que todas ações/estados sejam julgados uma boa quantidade de vezes).

O Q-Learning geralmente é implementado como duas matrizes de tamanhos $n \times n$ onde n é o número de estados.

Na matriz a seguir 3.1 é mostrado a matriz de recompensa ou **R** onde as linhas são os estados e as colunas são ações. Nesse exemplo a matriz é iniciada com 0, o estado objetivo (estado 5) é marcado como 100 e os estados que não podem ser alcançados são marcados com -1(por exemplo $R(0, 1) = -1$ quer dizer que é impossível ao estar no estado 0 ir ao

estado 1).

$$R_{n,n} = \begin{pmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{pmatrix} \tag{3.1}$$

A segunda matriz é a matriz de aprendizagem do Q-Learning chamada de **Q**. A matriz é iniciada com zero em todos os valores e durante o algoritmo esses valores serão ajustados.

Conhecidos as duas matrizes o procedimento de aprendizagem pode ser definido pela seguinte equação 3.2:

$$Q(s,a) = R(s,a) + \gamma \times Max[Q(s',a*)] \tag{3.2}$$

Onde:

- **s**. É o estado atual,
- **a**. É a ação escolhida,
- γ. É o fator de desconto ou taxa de aprendizagem. Segundo [Mitchell, 1997] esse valor pode ser qualquer constante entre $0 \leq \gamma \leq 1$. Se γ é próximo de zero, o agente tenderá a considerar mais recompensas imediatas, se próximo de 1 o agente considerará mais as futuras recompensas,
- **s'**. O próximo estado.
- **a***. Todas as ações do próximo estado.

O treinamento consiste em realizar diversas vezes essa equação até que a matriz Q convirja para valores sólidos.

3.4 Neuroevolução

Uma técnica bastante comum e bastante utilizada na pesquisa de inteligência para jogos é a neuroevolução. Segundo [Risi and Togelius, 2014] neuroevolução se refere a geração de redes neurais (pesos de suas conexões e/ou topologias) usando algoritmos evolutivos. Além de poder ser utilizada para um grande número de propósitos em jogos, os jogos são excelentes ambientes de testes para pesquisa de neuroevolução.

3.4.1 Redes Neurais

Segundo [Koehn, 1994] as redes neurais foram inventadas no espírito de ser uma metáfora biológica. A metáfora biológica para redes neurais é o cérebro humano. Como o cérebro, esse modelo computacional consiste em pequenas unidades interconectadas. Essas unidades (ou nós) têm habilidades bem simples. Assim, a força desse modelo deriva da interação dessas unidades. Ela depende da sua estrutura (topologia) e suas conexões.

Essas pequenas unidades, conexões e topologias podem ser comparadas a neurônios e sinapses. Um modelo bem comum e bastante utilizado é o chamado *perceptron*. Segundo [Beiu, 2003] uma rede neural de *perceptron* é composta por um grafo onde os nós são neurônios e as arestas são as sinapses. Essa rede tem alguns nós de entrada, e alguns (ao menos um) nós de saída.

O modelo simples de *perceptron* funciona com saídas binárias, sendo muito utilizado para reconhecimento de padrões, ele funciona da seguinte forma:

- Uma vetor de entrada X, onde cada posição do vetor representa uma característica diferente da entrada.
- Uma vetor W chamado de vetor de pesos sinápticos, para cada característica de entrada um peso é associado. Nesse vetor ocorrerão ajustes de valor até que a saída fique correta.
- Bias (b) é um valor de polarização que permite mudar a função de ativação para a esquerda ou para a direita, ou seja, ela permite melhor espalhar os resultados que a função de ativação gera esse valor também é ajustado durante o aprendizado.

A saída do *perceptron* pode ser definida pela seguinte equação:

$$y = \sigma(\sum_{j=1}^{n} w_j x_i j + b_i) \tag{3.3}$$

Onde:

- **y** é o valor de saída(0 ou 1).
- σ é a função de ativação. Retorna 1 para valores maiores que zero, e retorna 0 para valores menores ou iguais a zero.
- **n** é o tamanho do vetor de entrada.

Até que para todas as entradas, todas saídas estejam corretas o vetor W e o *bias* são atualizados. Segundo [Estebon, 1997] ajustando os pesos das conexões entre as camadas,

a saída do *perceptron* pode ser ”treinada”para corresponder com a saída desejada. O treino é completo quando um conjunto de entradas passa pela rede e os resultados obtidos são os resultados desejados. Se existir alguma diferença entre a saída atual e a saída desejada, os pesos são ajustados na camada de adaptação para produzir um conjunto de saída mais próximo aos valores desejados.

A equação de treinamento (ajuste de pesos) pode ser escrita da seguinte forma:

$$w'_ij = w_ij + \alpha(t_j - e_j) \times x_ij \tag{3.4}$$

Onde:

- w'_ij é o novo valor do peso.
- w_ij é o valor atual do peso.
- α taxa de aprendizagem onde $0 \leq \alpha \leq 1$. Valores baixos para α faz com que os pesos sejam ajustados suavemente.
- t_j Resultado esperado.
- e_j Resultado atual.
- x_ij Valor de entrada.

No algoritmo 2 é implementado o *Perceptron* aplicado para reconhecimento de classes famosas em jogos medievais (mago, ladino, guerreiro, clérigo). A entrada do algoritmo é uma matriz 4x6 representando os atributos força, destreza, constituição, inteligência, sabedoria e carisma. Os valores de atributos e definições de classes são do sistema *Dungeons & Dragons* [of the Coast, 2016].

3.4.2 Algoritmos genéticos

A primeira pesquisa na área de algoritmos genéticos foi feito por John Holland no livro *Adaptation in Natural and Artificial Systems*. Segundo [Mitchell, 1995] algoritmos genéticos(GA) é uma abstração da evolução biológica, onde move-se uma população de cromossomos (representando candidatos a soluções de um determinado problema) para uma nova população, usando ”seleção”junto com operadores baseados em genéticas para cruzamentos e mutação.

Segundo [Koza, 1995] antes de aplicar o algoritmo genético, é necessário mapear qual será a arquitetura do cromossomo de modo que ele represente uma solução para o problema. Para usar o modo convencional do algoritmo genético com cromossomos de tamanhos fixos, é preciso:

Algorithm 2 Algoritmo Perceptron

1: $input \leftarrow \begin{pmatrix} 10 & 16 & 12 & 16 & 13 & 8 \\ 8 & 16 & 10 & 13 & 12 & 16 \\ 16 & 9 & 15 & 13 & 11 & 14 \\ 14 & 8 & 15 & 10 & 16 & 12 \end{pmatrix}$ ▷ Input attributes

2: $out \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ ▷ Expected output

3: $L \leftarrow 0.1$ ▷ Learning Rate

4: **procedure** PERCEPTRON

5: $Neurons[4] \leftarrow initializeNeuronsRandomly()$

6: **while** !trainned **do**

7: $trainned \leftarrow true$

8: **for** each neuron $N \in Neurons$ **do**

9: $i \leftarrow 0$

10: **for** $j = 0$ to 4 **do** ▷ Number of classes

11: $total \leftarrow 0$

12: **for** $k = 0$ to 6 **do** ▷ Number of attributes

13: $total \leftarrow total + (input[j][k] \times N.weights[k] + N.bias)$

14: **end for**

15: **if** $total > 0$ **then**

16: $total \leftarrow 1$

17: **else**

18: $total \leftarrow 0$

19: **end if**

20: **if** $out[i][j] \neq total$ **then**

21: $trainned \leftarrow false$

22: **for** $k = 0$ to 6 **do** ▷ Updating Weights

23: $N.weights[k] \leftarrow N.weights[k] + L \times (out[j][k] - total) \times input[j][k]$

24: $N.bias \leftarrow N.bias + L \times (out[j][k] - total)$

25: **end for**

26: **end if**

27: **end for**

28: $i \leftarrow i + 1$

29: **end for**

30: **end while**

31: **end procedure**

- Determinar o esquema de representação.
- Determinar como mensurar o *fitness*. Segundo [Mitchell, 1995] o *fitness* é definido por uma função que atribui uma nota (*fitness*) para cada cromossomo da população atual. O *fitness* do cromossomo depende de quão bem o cromossomo resolve o problema.
- Determinar os parâmetros e variáveis para controlar o algoritmo.
- Determinar um modo de mostrar o resultado e um critério de parada para o algoritmo.

Segundo [Koza, 1995] O algoritmo evolutivo pode ser separados em três passos:

- Randomicamente criar uma população inicial de cromossomos (soluções para o problema).
- Executar os seguintes sub-passos na população até que o critério de parada seja satisfeito:

  1: Determinar para cada indivíduo um *fitness* através da função avaliadora de *fitness*.

  2: Selecionar quais indivíduos passarão para a próxima geração por uma probabilidade baseada em seu *fitness*. Aplicar na nova população de indivíduos as 3 seguintes operações genéticas: copiar um indivíduo para a nova população; criar dois novos indivíduos combinando seus cromossomos através de alguma operação de cruzamento (*crossover*); fazer a mutação de algumas características de um cromossomo aleatoriamente.
- Pegar o melhor indivíduo já criado (aquele com melhor *fitness*) para utilizar na resolução do problema.

O resultado de algoritmo genético muitas vezes não é o melhor resultado possível, porém gera um resultado excelente em cenários onde não se sabe como resolver um problema, ou em cenários onde a solução ótima demora muito tempo para ser calculada.

3.4.3 Neuroevolução

No trabalho *Combining genetic algorithms and neural networks: The encoding problem* de [Koehn, 1994] é levantado a seguinte questão: Se ambas técnicas são autônomas, porque então combiná-las? Ainda segundo [Koehn, 1994] a resposta curta para essa questão diz que o problema de redes neurais é o número de parâmetros que precisam ser atribuídos

antes de qualquer treino começar, nesse ponto entra o algoritmo genético. O autor do trabalho completa dizendo que a inspiração vem da natureza, o sucesso de um indivíduo não é determinado apenas pelo conhecimento e habilidades que ele ganha através da experiência, também depende de sua herança genética.

Existem também outros modos de aplicar algoritmos evolutivos em rede neurais, é comum encontrar abordagens onde o algoritmo genético é utilizado para fazer o treinamento de redes neurais. Segundo [Risi and Togelius, 2014], cada indivíduo é codificado em uma rede neural, e submetido para uma determinada tarefa por uma determinada quantidade de tempo. O desempenho da rede ou *fitness* é então guardado e uma vez que os valores de aptidão (*fitness*) para os genótipos (indivíduos) na população atual são determinados, uma nova população é gerada trocando as codificações do genótipo através de mutações e combinando os genótipos através de cruzamentos. Em geral, genótipos com alto *fitness* tem uma alta chance de ser selecionado para reprodução e os seus descendentes substituem genótipos com valores de *fitness* mais baixos, formando assim uma nova geração.

No trabalho de [Moriarty and Miikkulainen, 1994] foi criado uma rede neuroevolutiva para jogar Reversi (jogo de tabuleiro), para isso, foi implementado uma população de 50 rede neurais evoluindo por 1000 gerações, para calcular o *fitness* de cada rede neural a rede era submetida a 244 jogos contra um agente baseado em *minimax*, a percentagem de vitória define o *fitness* do indivíduo.

Capítulo 4

Sistemas de batalhas Pokémon

Esse capítulo apresentará o sistemas de batalhas dos jogos *Pokémon*. Como dito nos capítulos anteriores é preciso conhecer bem o ambiente que o agente está inserido para poder aferir avaliações e tomar decisões. Este capítulo está organizado da seguinte maneira: na seção 4.1 é feita uma breve contextualização sobre o jogo e a franquia *Pokémon*. Na seção 4.2 é apresentado o simulador *on-line* de batalhas chamado de *Pokémon Showdown!*. No capítulo 4.3 será explicado como funciona o sistema de batalha.

4.1 Contextualização

Segundo [Panumate et al., 2015] *Pocket Monsters* ou Pokémon é uma série de jogos desenvolvida pela *Game Freak and Creatures Inc.* e publicada pela Nintendo como parte da franquia Pokémon. O primeiro lançamento foi em 1996 no Japão para o console Game Boy Color, a principal série do jogo é baseada em *Role-Playing-Game* (RPG) e continua a ser lançado para consoles portáteis até hoje.

Ainda segundo [Panumate et al., 2015] o objetivo dos jogos de RPG de Pokémon é capturar monstros chamados de Pokémon e com eles conseguir ganhar insígnias de ginásios (prêmio dado por vencer o líder de ginásio em uma batalha Pokémon), para assim ter acesso a liga Pokémon e se tornar o campeão dessa (torneio com os melhores treinadores Pokémon do jogo).

4.2 Pokémon Showdown!

Pokémon Showdown é um simulador de batalhas Pokémon. Permitindo jogar batalhas *online*! Jogar com times gerados aleatoriamente, ou construindo seu próprio time ([Showdown!, 2016]).

Esse ambiente é totalmente *on-line* e pode ser jogado via navegador de internet. O código fonte do jogo é disponibilizado sobre a licença MIT. O dono e principal mantenedor do projeto é Guangcong Luo e todo código fonte do projeto pode ser encontrado no Git Hub.

O ambiente conta um grande número de jogadores simultâneos. Durante o desenvolvimento do trabalho foi verificado que muitas vezes o número de usuários simultâneos ultrapassou a quantidade de 10.000.

O modo de batalha utilizado durante esse trabalho é o *Random battle* onde o jogo procura um jogador oponente para batalhar e dá para ambos, times aleatórios.

4.3 Batalhas

Para poder entender como funciona a mecânica dos jogos Pokémon é preciso apresentar todos os elementos que fazem parte do sistema de batalha.

4.3.1 Características Pokémon

Atualmente existem mais de 720 espécies de Pokémons diferentes disponíveis no jogo. As principais características dos Pokémons são:

- **Espécie**. Nome do Pokémon (Bulbasaur, Charmander, Squirtle e etc);
- **Tipo**. Um Pokémon pode ter um ou dois tipos diferentes;
- **Gênero**. Um Pokémon pode macho, fêmea ou sem gênero;
- **Peso**. Peso do Pokémon;
- **Estatísticas base**. As estatística ou atributos são separado em 6 categorias:

  HP ou PV Quantidade de pontos de vida Pokémon;

  Attack ou Ataque Quantidade de ataque físico do Pokémon;

  Defense ou Defesa Quantidade de defesa física do Pokémon;

  Special Attack ou Ataque Especial Quantidade de ataque mágico do Pokémon;

  Special Deffense ou Defesa Especial Quantidade de defesa mágica do Pokémon;

  Speed ou Velocidade Velocidade do Pokémon;

- **Habilidades**. Um Pokémon tem uma habilidade especial (muitas vezes o que diferencia 2 Pokémons do mesmo tipo além das estatísticas base, são suas possíveis habilidades);

Para calcular a estatística real do Pokémon alguns outros atributos são necessários. Esses atributos são definidos pelo jogador (com exceção do nível que é definido pelo Pokémon Showdown em alguns modos de jogo):

- **Level**. Nível do Pokémon. Onde $0 \leq level \leq 100$.
- **Effort Values ou EV**. Valores de esforço, cada Pokémon tem 510 pontos de EV que podem ser distribuídos em qualquer um dos 6 atributos, com a limitação de que cada atributo pode ter no máximo 252 pontos de EV.
- **Individual Values ou IV**. Valores individuais, cada estatística do Pokémon tem um valor de IV associado que varia entre 0 e 31. No ambiente Pokémon Showdown todos os Pokémons estão com 31 de IV para todos os 6 atributos.
- **Nature**. Natureza do Pokémon. Existem 25 diferentes naturezas que um Pokémon pode ter, sendo que 21 delas alteram atributos. As naturezas que alteram atributos adicionam 10% em um determinado atributo e reduzem 10% em outro atributo. A tabela 4.1 mostra as possíveis naturezas e as modificações dos atributos.

Finalmente o cálculo da estatística atual é dado pela seguinte equação:

$$V = \left(\frac{\left((2 \times Base + IV + \frac{EV}{4}) \times level \right)}{100} + 5 \right) \times Nature \tag{4.1}$$

Essa equação vale para todos os atributos menos para o atributo HP. A Equação do HP tem uma pequena modificação que é mostrada na equação a seguir:

$$V = \left(\frac{\left((2 \times Base + IV + \frac{EV}{4}) \times level \right)}{100} + level + 10 \right) \tag{4.2}$$

Onde:

- **V** É o valor final, valor que o jogador irá ver no jogo.
- **Base** É a quantidade básica da estatística.
- **Nature** Se a natureza do Pokémon favorecer esse atributo o valor é atribuído 1.1, caso a natureza desfavoreça esse valor é 0.9 e caso a natureza não influa nesse atributo o valor é 1.

Tabela 4.1: Naturezas do Pokémon

Natureza	**Atributo aumentado**	**Atributo diminuído**
Lonely	Ataque	Defesa
Brave	Ataque	Velocidade
Adamant	Ataque	Ataque Especial
Naughty	Ataque	Defesa Especial
Bold	Defesa	Ataque
Relaxed	Defesa	Velocidade
Impish	Defesa	Ataque Especial
Lax	Defesa	Defesa Especial
Modest	Ataque Especial	Ataque
Mild	Ataque Especial	Defesa
Quiet	Ataque Especial	Velocidade
Rash	Ataque Especial	Defesa Especial
Calm	Defesa Especial	Ataque
Gentle	Defesa Especial	Defesa
Sassy	Defesa Especial	Velocidade
Careful	Defesa Especial	Ataque Especial
Timid	Velocidade	Ataque
Hasty	Velocidade	Defesa
Jolly	Velocidade	Ataque Especial
Naive	Velocidade	Defesa Especial
Hardy	Nenhum	Nenhum
Docile	Nenhum	Nenhum
Serious	Nenhum	Nenhum
Bashful	Nenhum	Nenhum
Quirky	Nenhum	Nenhum

EVs, IVs e Natureza do Pokémon são importantes customizações que fazem um Pokémon ser diferente de outro, mesmo eles sendo da mesma espécie. Na batalha esses 3 valores não estão disponíveis para o seu adversário.

Para exemplificar e mostrar a diferença que a distribuição de valores pode causar, a seguir será calculado o atributo "Especial Ataque"para 2 Pokémons da espécie Mewtwo ambos no nível 100. O Pokémon Mewtwo tem 154 pontos de Ataque Especial base.

O primeiro Mewtwo será chamado de Mewtwo A. O Mewtwo A é da natureza Modest e tem 252 pontos de EV em Ataque Especial e 31 pontos de IV no atributo Ataque Especial. Substituindo essas variáveis na equação:

$$447.7 = \left(\frac{\left((2 \times 154 + 31 + \frac{252}{4}) \times 100 \right)}{100} + 5 \right) \times 1.1 \tag{4.3}$$

O segundo Mewtwo será chamado de Mewtwo B. O Mewtwo B é da natureza Adamant e tem 0 pontos de EV em Ataque Especial e 0 pontos de IV no atributo Ataque Especial. Substituindo essas variáveis na equação:

$$281.7 = \left(\frac{\left((2 \times 154 + 0 + \frac{0}{4}) \times 100 \right)}{100} + 5 \right) \times 0.9 \tag{4.4}$$

A diferença do atributo Especial Ataque do Mewtwo A e do Mewtwo B é de 166 pontos, ou seja, o Mewtwo A tem cerca de 66% mais pontos nesse atributo que o Mewtwo B.

Habilidades, itens e golpes dos Pokémons

Esses 3 aspectos são importantes na customização do Pokémon, pois além de ter uma grande variedade de opções, tais aspectos não são visíveis para o seu oponente.

Habilidades são efeitos passivos que podem ajudar dentro de batalha. Pokémons tem entre 1 e 3 possíveis habilidades podendo escolher apenas 1 delas (ficando a cargo do jogador qual escolher). Existem cerca de 200 habilidades diferentes. Um exemplo de habilidade é a *Intimidate*, quando um Pokémon com a habilidade *Intimidate* entra em batalha o Pokémon adversário tem seu ataque reduzido em 50%.

Cada Pokémon pode carregar consigo um item, esse item também traz certos tipos de benefícios para o Pokémon. Alguns itens são ativados quando uma condição ocorre na batalha, outros itens são contínuos. Existem cerca de 300 itens diferentes. Um exemplo de item é o *Leftovers* que recupera $\frac{1}{16}$ do HP total do Pokémon ao final de cada turno.

Uma das customizações mais importantes é quais golpes o Pokémon irá poder utilizar. Cada Pokémon pode escolher 4 golpes entre dezenas de golpes disponíveis. Cada espécie de Pokémon pode aprender um conjunto diferente de habilidades dependendo do seu tipo e características (por exemplo, apenas Pokémons com chifre conseguem aprender o golpe Mega Horn). Existem cerca de 600 golpes diferentes.

As principais características dos golpes são:

- **Base Power ou Base de Força**. A força base utilizada no cálculo de dano.
- **Category ou Categoria**. Existem três tipos de categorias:

  Status ou Condição Golpes que não infringem dano diretamente, porém tem algum efeito com grande chance de sucesso para compensar.

  Physical ou Físico Golpes físicos utilizam o Ataque para cálculo de dano e Defesa para cálculo da resistência.

  Special ou Especial Golpes especiais utilizam o Ataque Especial para cálculo de dano e Defesa Especial para cálculo da resistência.

- **Tipo**. Se o tipo do golpe for um dos tipos do Pokémon a base de força é aumentada em 50% (fenômeno chamado de STAB ou *Same Type Attack Bonus*).
- **Accuracy ou Precisão**. A chance em percentagem de um golpe acertar.
- **PP ou Pontos de Poder**. A quantidade de vezes que um Pokémon pode usar um golpe durante uma batalha.
- **Efeito Secundário**. Um golpe pode ter um efeito secundário, esse efeito tem uma chance de acontecer e pode afetar uma série de coisas como, por exemplo, deixar o oponente paralisado ou aumentar algum atributo.

4.3.2 Composição dos times

Os dois modos de batalhas mais jogados do Pokémon Showdown são: Random Battles e OU Battles. Em ambos os modos cada time conta com 6 Pokémons (grande maioria dos tipos de jogos utilizam 6 Pokémons para compor um time).

Para entender melhor esses 2 modos é preciso entender algumas definições do Pokémon competitivo.

Existe uma comunidade chamada Smogon [Smogon, 2016], essa comunidade regula e define as regras que são utilizadas em cada modo de jogo, para que o jogo se torne mais competitivo e balanceado. Um dos principais trabalhos da Smogon é balancear os mais

de 720 Pokémons distribuindo eles em faixas chamadas de *tiers*. Do mais fraco para o mais fortes os principais *tiers* são:

- *Never Used* ou NU;
- *Rarely Used* ou RU;
- *Under Used* ou UU;
- *Over Used* ou OU;
- *Ubers*.

No modo OU Battles como o próprio nome sugere o jogador pode montar seu time com qualquer Pokémon do *tier* OU e de *tiers* inferiores, nesse modo todos os Pokémons se enfrentarão no mesmo nível.

Em batalhas randômicas, para balancear os Pokémons de diferentes *tiers* é atribuído diferentes níveis de acordo com o *tier* do Pokémon. Pokémons NU batalham no nível 86, RU 82, UU 78, OU 74, *Ubers* 70.

Em cada diferente modo de jogo o jogador tem sua pontuação de **Elo** que é uma pontuação no ranquemanto geral do jogo. Em cada jogo o jogador ganha ou perde pontos de Elo de acordo com o resultado da partida. A quantidade de pontuação ganha é feita pela diferença do seu Elo para o adversário. O jogo sempre tenta parear jogadores com pontuação de Elo semelhante.

Por padrão cada jogador inicia com 1000 pontos de Elo. Em batalhas com níveis parecidos de Elo é comum ganhar/perder 20 pontos por partida. Atualmente os jogadores que lideram as colocações do modo Random Battle têm cerca de 2.200 pontos.

4.3.3 Sistema de batalhas

Nessa subseção serão apresentados os principais aspectos de uma batalha, estes aspectos são vitais para construção dos agentes, pois é baseado nesses pontos que as decisões serão tomadas.

O sistema de batalhas é baseado em dois jogadores em um sistema de turno, ou seja, a cada começo de turno, cada jogador escolhe qual ação irá tomar.

No modo Random Battles um de seus Pokémons é escolhido aleatoriamente para começar a batalha. Em modos como OU Battles é possível escolher qual será o primeiro Pokémon em campo. Os Pokémons colocados em primeiro na batalha são chamados de *lead* e são uma peça muito importante para o time.

Em situações normais, cada turno o jogador pode escolher duas grandes ações. A primeira é utilizar um dos quatro golpes do Pokémon. A segunda é trocar o Pokémon em campo para outro Pokémon desde que ele esteja vivo. Ao trocar de Pokémon você perde o seu turno, ou seja, logo que o Pokémon escolhido entrar em batalha ele tomará o golpe que o adversário escolheu no início de seu turno (caso o oponente não tenha escolhida trocar de Pokémon).

Uma das primeiras situações onde os atributos do Pokémon são relevantes é para definir quem será o primeiro Pokémon a atacar. Em situações normais o primeiro Pokémon que ataca é aquele com Velocidade entre os dois.

Sistema de vantagens de tipos

Alguns tipos de Pokémons tem vantagem sobre outros tipos de Pokémons. A ideia é baseada no sistema papel pedra tesoura, onde o papel ganha de pedra, pedra ganha de tesoura e por sua vez tesoura ganha de papel.

Porém no sistema de batalhas Pokémon essa mecânica é um pouco mais complexa. Além de existir 18 tipos diferentes de Pokémon, um Pokémon pode ter dois tipos distintos.

A imagem a seguir foi retirada do site [Bulbapedia, 2016] e contém uma tabela mostrando o modificador de dano para cada tipo de golpe em relação ao tipo do adversário. Ao escolher um golpe, o tipo (golpes tem apenas um único tipo) desse golpe é confrontado nessa tabela de acordo com o tipo do adversário, caso o adversário tenha dois tipos a consulta nessa tabela é feita duas vezes e os modificadores são multiplicados gerando um modificar final.

Sistema de cálculo de dano

Para calcular a quantidade de HP que um Pokémon irá perder é utilizada a seguinte equação:

$$Dano = \left(\frac{2 \times Level + 10}{250} \times \frac{Attack}{Defense} \times Base + 2 \right) \times Modifier \tag{4.5}$$

Onde:

- **Dano**. Quantidade de dano que o Pokémon irá receber.
- **Attack**. É quantidade de Ataque caso o golpe seja da categoria físico, ou quantidade Ataque Especial caso o golpe seja da categoria especial.

Attacking type \ Defending type	NORMAL	FIGHT	FLYING	POISON	GROUND	ROCK	BUG	GHOST	STEEL	FIRE	WATER	GRASS	ELECTR	PSYCHC	ICE	DRAGON	DARK	FAIRY
NORMAL	1×	1×	1×	1×	1×	½×	1×	0×	½×	1×	1×	1×	1×	1×	1×	1×	1×	1×
FIGHT	2×	1×	½×	½×	1×	2×	½×	0×	2×	1×	1×	1×	1×	½×	2×	1×	2×	½×
FLYING	1×	2×	1×	1×	1×	½×	2×	1×	½×	1×	1×	2×	½×	1×	1×	1×	1×	1×
POISON	1×	1×	1×	½×	½×	½×	1×	½×	0×	1×	1×	2×	1×	1×	1×	1×	1×	2×
GROUND	1×	1×	0×	2×	1×	2×	½×	1×	2×	2×	1×	½×	2×	1×	1×	1×	1×	1×
ROCK	1×	½×	2×	1×	½×	1×	2×	1×	½×	2×	1×	1×	1×	1×	2×	1×	1×	1×
BUG	1×	½×	½×	½×	1×	1×	1×	½×	½×	½×	1×	2×	1×	2×	1×	1×	2×	½×
GHOST	0×	1×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	1×
STEEL	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×	½×	1×	½×	1×	2×	1×	1×	2×
FIRE	1×	1×	1×	1×	1×	½×	2×	1×	2×	½×	½×	2×	1×	1×	2×	½×	1×	1×
WATER	1×	1×	1×	1×	2×	2×	1×	1×	1×	2×	½×	½×	1×	1×	1×	½×	1×	1×
GRASS	1×	1×	½×	½×	2×	2×	½×	1×	½×	½×	2×	½×	1×	1×	1×	½×	1×	1×
ELECTR	1×	1×	2×	1×	0×	1×	1×	1×	1×	1×	2×	½×	½×	1×	1×	½×	1×	1×
PSYCHC	1×	2×	1×	2×	1×	1×	1×	1×	½×	1×	1×	1×	1×	½×	1×	1×	0×	1×
ICE	1×	1×	2×	1×	2×	1×	1×	1×	½×	½×	½×	2×	1×	1×	½×	2×	1×	1×
DRAGON	1×	1×	1×	1×	1×	1×	1×	1×	½×	1×	1×	1×	1×	1×	1×	2×	1×	0×
DARK	1×	½×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×
FAIRY	1×	2×	1×	½×	1×	1×	1×	1×	½×	½×	1×	1×	1×	1×	1×	2×	2×	1×

These matchups are suitable for Generation VI.

Figura 4.1: Tabela de vantagens e desvantagens do Pokémon. [Bulbapedia, 2016]

- **Denfese**. É quantidade de Defesa caso o golpe seja da categoria físico, ou quantidade Defesa Espcial caso o golpe seja da categoria especial.
- **Base**. Base de força definido no golpe.
- **Modifier**. É um conjunto de modificadores que são aplicados ao causar dano. Esses modificadores podem ser definidos pela seguinte equação:

$$Modifier = STAB \times Type \times Critical \times other \times (random[0.85, 1]) \tag{4.6}$$

Onde:

- **STAB** É um bônus que é habilitado quando o golpe tem o mesmo tipo do Pokémon.
- **Type** É o modificador baseado na vantagem, explicado na subseção anterior, os valores podem variar entre 0, 0.25, 0.5, 1, 2, e 4.
- **Critical** Normalmente um golpe tem chance de crítico de 6.25% caso esse acerto crítico ocorra o valor para Critical é de 1.5, caso contrário o valor é 1.

- **other** São outros modificadores que podem ser adicionados por causa de itens ou estados especiais que modifiquem o dano.
- **random** Por final é adicionado um valor aleatório entre 85% e 100%.

Estados especiais de batalha

Existem estados de batalha que pode mudar bastante a tomada de decisão de um agente, pois esses estados modificam bastante o ritmo da batalha.

Os principais estados de batalhas que o Pokémon pode ser submetido são chamados de Estados primários. Os Pokémons só podem ser afetados por apenas 1 estado primário de batalha, ou seja, caso tente ser aplicado um estado de batalha em um Pokémon que já está sofrendo um estado de batalha esse novo estado falhará. Os principais estados são:

- **BRN** Estado queimado, nesse estado o Ataque do Pokémon é reduzido em 50% e, ao final de cada turno o Pokémon recebe 12,5% da vida máxima como dano.
- **FRZ** Estado congelado, nesse estado o Pokémon não pode utilizar nenhum golpe, a cada turno o Pokémon tem uma chance de 20% de se descongelar.
- **PAR** Estado paralisado, nesse estado o Pokémon tem 25% de chance de errar seu golpe e, sua Velocidade é reduzido para 25% do valor.
- **PSN** Estado envenenado, nesse estado o Pokémon perde $\frac{1}{16}$ de seu HP no final do turno, aumentando em $\frac{1}{16}$ para cada turno que ele continuar envenenado.
- **SLP** Estado dormindo, nesse estado o Pokémon não pode atacar. Esse efeito tem duração de 1 até 3 turnos.

Existem outras categorias de efeitos que são chamados de efeitos secundários ou efeitos voláteis. Diferentemente dos estados primários, um Pokémon pode ter N desses efeitos aplicados. Existe uma grande lista de efeitos secundários. Um efeito secundário bastante conhecido é a confusão, onde o Pokémon fica confuso de 1 a 4 turnos, caso o Pokémon esteja confuso ele tem 50% de chance de bater em si próprio ao invés de usar o golpe no inimigo.

Final de Batalha

Uma batalha normalmente acaba quando todos os Pokémons de um jogador estão desmaiados, ou seja, estão com 0 de HP.

Uma batalha do Pokémon Showdown pode acabar também por desistência do jogador ou por desconexão de um dos jogadores.

Capítulo 5

Metodologia

Para atingir os objetivos propostos serão implementados três agentes. Dois desses agentes serão submetidos a uma grande quantidade de partidas de treinamento para que possam chegar a patamares estáveis. O terceiro será um agente baseado em grafos com diferentes profundidades, onde será categorizada a evolução no ranquemanto do jogo de acordo com o aumento de profundidade da árvore de decisão.

- **Agente 1 com neuroevolução**. O primeiro agente será uma rede neural onde os pesos de sua rede serão ajustados por um algoritmo evolutivo.
- **Agente 2 com aprendizado por reforço**. Esse agente será ajustado através de estímulos positivos e negativos que regularão sua decisão dentro do jogo através do algoritmo de aprendizado por reforço.
- **Agente 3 baseado em árvore de decisão**. Utilizará uma árvore de decisão (algoritmo baseado em *minimax*) onde os possíveis movimentos serão mapeados. O algoritmo tentará prever possíveis incógnitas do adversário assumindo o pior cenário possível para cada variável não conhecida.

5.1 Treino e aprendizado

O primeiro agente que será testado contra jogadores humanos é o agente 3 (árvore). Serão criadas diferentes versões desse agente. Cada versão analisará diferentes profundidades de árvores, ou seja, serão criados N versões do agente, a primeira versão analisará jogadas até a x profundidade da árvore, a próxima versão analisará até x + 1. O jogo tem um limite máximo de dois minutos para fazer sua jogada, com base nesse limite de tempo serão reguladas quantas diferentes versões dessa técnica serão criadas.

Para executar essa abordagem será criado uma função de avaliação de heurística baseado nas informações da seção 4.3. Essa técnica será importante para evitar problema de grande tempo de execução. Além disso, poderemos confrontar os outros dois agentes com diferentes versões desse agente para comparar o nível de aprendizado de cada agente.

Os agentes 1 e 2 terão comportamentos semelhantes em como serão treinados. Ambos agentes terão duas versões, cada versão será submetida aos seguintes treinamentos:

- **Treino contra humanos**: O agente será submetido a jogar contra jogadores humanos que serão escolhidos pelo próprio jogo, o sistema de pareamento de partidas é feito baseado no ranqueamento dos dois jogadores, ou seja, o adversário sempre vai ter uma pontuação no *rank* semelhante ao agente.
- **Treino contra agente 3**: Os agentes 1 e 2 jogarão contra o agente 3, sempre que o agente que está treinando conseguir um grande números de percentagem de vitória sobre o agente 3, será aumentado mais um nível de profundidade na árvore do agente 3.

5.2 Avaliação de resultado

Avaliar a situação da batalha será um recurso muito importante, pois o agente 3 precisará avaliar cada nó de sua árvore para fazer a melhor escolha possível, ou a escolha que gere menos boas escolhas para o adversário. Essa avaliação terá como base a quantidade de Pokémons vivos e a situação deles (quantidade de pontos de vida, afetado por algum efeito negativo entre outros).

Ao fim de cada batalha também será feita uma avaliação do jogo. Com essa avaliação será possível aferir o quão díspar foi o resultado para o vencedor, além disso, com essa métrica podemos eliminar possíveis ruídos em resultados de batalhas como desistências ou desconexões por parte dos oponentes.

A heurística escolhida para definir uma pontuação para o estado atual é baseada na soma da multiplicação da porcentagem de pontos de vida atual vezes os modificadores de batalha de cada Pokémon (o capítulo 4 mostra mais detalhes sobre cada variável). A diferença de pontos dos dois times será a heurística adotada. A pontuação de cada time é definida pela seguinte equação:

$$\sum_{i=1}^{6} HP_i x \times Mod_i \tag{5.1}$$

Onde:

- **i**: Cada Pokémon.
- **HP**: Percentagem restante de pontos de vida.
- **Mod**: Modificadores que podem melhorar (e.g. dobro de ataque, 1.5 vezes a velocidade, etc.) ou piorar (e.g. metade do ataque, 66% de precisão, estados especiais de batalha explicadas na seção 4.3.3).

Capítulo 6

Plano de Trabalho

As atividades compreendidas no cronograma são:

- **Revisão de literatura (RDL)**: Leitura e entendimento de trabalhos que permeiam o assunto da dissertação dentre eles: redes neurais, computação evolutiva, neuroevolução, aprendizado por reforço, Monte Carlo, agentes inteligentes para jogos entre outros.

- **Estudo de caso AIBirds (AIB)**: Criação de um agente inteligente baseado em árvore para a competição AIBirds. AIBirds ou *Angry Birds AI Competition* é uma competição entre agentes que joguem o jogo Angry Birds. Segundo sua página oficial "A tarefa dessa competição é desenvolver um programa de computador que jogue Angry Birds autonomamente e sem intervenção humana."[AIBirds, 2016] A competição foi sediada no congresso IJCAI 2015 e o agente desenvolvido foi nomeado UFAngryBirdsC. O desenvolvimento desse agente teve grande importância para o trabalho, além da experiência de criação de agentes serviu como inspiração para o tema do trabalho.

- **Desenvolvimento de API de comunicação com o jogo Pokémon Showdown (API)**: Para que seja possível criação de agentes foi desenvolvida uma API em JavaScript que recuperar todas informações e pode realizar as mesmas ações que um jogador humano pode fazer. Durante o desenvolvimento foi tomado muito cuidado para que a API não tenha mais informações ou ações que um jogador comum pode obter ou realizar.

- **Escrever documento de qualificação (EDQ)**: Escrever o documento para o exame de qualificação.

- **Desenvolvimento dos três agentes (DTA)**: Como já citado anteriormente serão criados três agentes utilizando técnicas diferentes como: reuroevolução, aprendizado

por reforço e algoritmo de grafos baseado em *minimax*.

- **Treino inicial dos Agentes (TIA)**: Como descrito na seção 5.1 será inicialmente treinado apenas o agente 3. Esse agente será muito importante para o treino dos próximos agentes e para sugerir uma categorização para jogadores humanos para cada faixa de ranqueamento.
- **Treino dos demais agentes (TDA)**: Treino dos agentes 1 e 2 contra o agente 3 e contra jogadores humanos. Nessa fase também será sumarizada a subida no ranqueamento de acordo com a quantidade de treino.
- **Compilação de resultados (CDR)**: Compilação dos resultados de cada agente e comparação da performance de cada agentes com jogadores humanos.
- **Escrever documento da dissertação (EDD)**: Escrever o documento para a defesa da dissertação.
- **Elaboração de artigo (EDA)**: Escrever um artigo com o tema da dissertação.
- **Desenvolver WebSocket de comunicação com outras linguagens (DWS)**: Será desenvolvido um WebSocket na API para que possa ser desenvolvido agentes em qualquer outra linguagem que tenha o recurso de WebSocket.
- **Escrever documentação da API (WIK)**: Fazer uma página mostrando como utilizar os principais recursos da API e como utilizar ela em outra linguagens através do WebSocket.

O cronograma abaixo compreende o período de maio de 2015 data onde foram feitos as primeiras etapas relacionadas a dissertação, até setembro de 2016 mês que pretende-se defender a dissertação.

Tabela 6.1: Cronograma da Dissertação

	2015								**2016**								
	05	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**
RDL	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
AIB	X	X	X														
API							X	X	X								
EDQ									X	X							
DTA										X	X	X	X	X			
TIA												X	X				
TDA													X	X			
CDR														X	X		
EDD															X	X	X
EDA														X	X		
DWS													X	X			
WIK														X	X		

Referências Bibliográficas

[aend Edward Powley et al., 2012] aend Edward Powley, C. B., Whitehouse, D., Lucas, S., Cowling, P. I., Tavener, S., Perez, D., Samothrakis, S., Colton, S., and et al. (2012). A survey of monte carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI*.

[AIBirds, 2016] AIBirds (2016). AIBirds, Angry Birds AI Competition. https://aibirds.org/. Acessado em: 01-02-2016.

[Beiu, 2003] Beiu, V. (2003). A survey of perceptron circuit complexity results. In *Intl. Joint Conf. Neural Networks IJCNN 03*, volume 2, pages 989–994.

[Bulbapedia, 2016] Bulbapedia (2016). Bulbapedia type chart. http://bulbapedia.bulbagarden.net/wiki/Type. Acessado em: 01-02-2016.

[Campbell and Marsland, 1983] Campbell, M. S. and Marsland, T. A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20(4):347–367.

[Christensen and Korf, 1986] Christensen, J. and Korf, R. E. (1986). A unified theory of heuristic evaluation functions and its application to learning. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, pages 148–152.

[CIG, 2016] CIG (2016). CIG 2014, ieee Conference on Computational Intelligence and Games. http://www.cig2014.de/doku.php. Acessado em: 08-02-2016.

[Doran and Parberry, 2011] Doran, J. and Parberry, I. (2011). A prototype quest generator based on a structural analysis of quests from four mmorpgs. In *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*, PCGames '11, pages 1:1–1:8, New York, NY, USA. ACM.

[Estebon, 1997] Estebon, M. D. (1997). Perceptrons: An associative learning network.

[Franklin and Graesser, 1997] Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on*

Intelligent Agents III, Agent Theories, Architectures, and Languages, ECAI '96, pages 21–35, London, UK, UK.

[GameIndustry, 2016] GameIndustry (2016). Gameindustry. http://www.gamesindustry.biz/. Acessado em: 01-02-2016.

[Hendrikx et al., 2013] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22.

[Herbrich, 2011] Herbrich, R. (2011). Drivatar - Microsoft Research. http://research.microsoft.com/en-us/projects/drivatar/default.aspx. Acessado em: 07-02-2016.

[Hsu et al., 1995] Hsu, F., Campbell, M.S., H., and A.J.J (1995). Deep blue system overview. *Proceedings of the 9th ACM Int. Conf. on Supercomputing*, pages 240–244.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285.

[Knuth and Moore, 1976] Knuth, D. E. and Moore, R. W. (1976). An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326.

[Koehn, 1994] Koehn, P. (1994). Combining genetic algorithms and neural networks: The encoding problem.

[Koza, 1995] Koza, J. R. (1995). Survey of genetic algorithms and genetic programming. In *WESCON/'95. Conference record. 'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*, pages 589–.

[Michaelis, 2016] Michaelis (2016). Michaelis, Dicionários Michaelis. http://michaelis.uol.com.br/. Acessado em: 07-02-2016.

[Millington and Funge, 2009] Millington, I. and Funge, J. (2009). *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.

[Mitchell, 1995] Mitchell, M. (1995). Genetic algorithms: An overview. *Complexity*, 1(1):31–39.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Moriarty and Miikkulainen, 1994] Moriarty, D. and Miikkulainen, R. (1994). Improving game-tree search with evolutionary neural networks. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 496–501 vol.1.

[Nielsen, 1994] Nielsen, J. (1994). Usability inspection methods. chapter Heuristic Evaluation, pages 25–62. John Wiley & Sons, Inc., New York, NY, USA.

[Nwana, 1996] Nwana, H. S. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11:205–244.

[of the Coast, 2016] of the Coast, W. (2016). Dungeons & dragons character sheets. http://dnd.wizards.com/articles/features/character_sheets. Acessado em: 01-03-2016.

[Ontanon et al., 2013] Ontanon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(4):293–311.

[Panumate et al., 2015] Panumate, C., Xiong, S., and Iida, H. (2015). An approach to quantifying pokemon's entertainment impact with focus on battle. In *Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI), 2015 3rd International Conference on*, pages 60–66.

[Ponsen and Spronck, 2004] Ponsen, M. and Spronck, I. P. H. M. (2004). Improving adaptive game ai with evolutionary learning. In *University of Wolverhampton*, pages 389–396.

[PWC, 2016] PWC (2016). Pwc. http://www.pwc.com/. Acessado em: 01-02-2016.

[Risi and Togelius, 2014] Risi, S. and Togelius, J. (2014). Neuroevolution in games: State of the art and open challenges.

[Russell and Norvig, 2010] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Discovering great minds of science. Prentice Hall.

[Schaeffer and van den Herik, 2002] Schaeffer, J. and van den Herik, H. (2002). Games, computers and artificial intelligence. *Artificial Intelligence - Chips challenging champions: games, computers and Artificial Intelligence*, 134:1–8.

[Showdown!, 2016] Showdown! (2016). Pokémon Showdown! http://pokemonshowdown.com/. Acessado em: 01-02-2016.

[Smith et al., 1994] Smith, D. C., Cypher, A., and Spohrer, J. (1994). Kidsim: Programming agents without a programming language. *Commun. ACM*, 37(7):54–67.

[Smogon, 2016] Smogon (2016). Smogon. http://www.smogon.com/. Acessado em: 01-02-2016.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[van Lent et al., 1999] van Lent, M., Laird, J. E., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K., and Tedrake, R. (1999). Intelligent agents in computer games. In Hendler, J. and Subramanian, D., editors, *AAAI/IAAI*, pages 929–930. AAAI Press / The MIT Press.

[Works, 2016] Works, H. D. B. (2016). How Deep Blue works. https://www.research.ibm.com/deepblue/meet/html/d.3.2.shtml. Acessado em: 07-02-2016.

[Yannakakis and Togelius, 2015] Yannakakis, G. and Togelius, J. (2015). A panorama of artificial and computational intelligence in games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 7(4):317–335.