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.C

Prof.Dr.B

Prof.Dr.D

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 sitema de ranqueamento do jogo Pokémon Showdown.

Escolher o melhor algoritmo e a melhor forma de aprendizado para esse agente 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 composições de equipes são todas aleatórias e a avaliação de performance do agente será feita através do sistema de ranqueamento do próprio jogo.

Abstract

To Do.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o grande crescimento da inteligência artificial e inteligência computacional e o estabelecimento como área de esquisa um grande número de trabalhos sobre criações de agentes, ou seja, *softwares* que fazem uma determinada tarefa utilizando alguma(s) estratégia(s) (no capítulo 2 será aprofundado o conceito de agentes) foi produzido. Otimização de treinos, busca em árvore de decisão e competição entre agentes são apenas alguns pontos do que está sendo pesquisado.

Para fomentar ainda mais essa área, um grande número de competições acadêmicas foi criado. Essas competições tem diferentes finalidades como batlalhas entre agentes, agentes que se comportam como jogadores humanos entre outros.

Esse trabalho irá explorar o ambiente de simulação de batalhas *Pokémon on-line* chamado de *Pokémon Showdown*. Para se comunicar com esse ambiente será feito uma API Application Programming Interface) onde serão desenvolvidos agentes para jogar contra jogadores humanos.

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 e testar algoritmos de aprendizados para agentes entre outros.

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* nada 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 montado de uma infinidade de modos diferentes. Para acentuar ainda mais essas proprieaddes os agentes irão treinar e competir no modo randômico, nesse modo a escolha do seu time assim como 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, assim 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. Será implementada distintas técnicas para criação e aprendizado desses agentes. No decorrer do trabalho 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 JavaScript mas durante o desenvolvimento do projeto será portada para Java através 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 *Pokémon* tem serão apenas conhecidos a medida que o adversário utilizá-los e existem caractéristicas que agente não tem como descobrir como por exemplo a quantidade de ataque e defesa distruibida no monstro.

Além disso a construção de uma API para acesso ao jogo contrubuirá 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 utilzar-se do termo agente ou agente inteligente. Por definição do dicionário [mic, 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 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 suas próprais 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, ao longo do tempo, em busca de sua própria agenda e 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 é 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 como agentes fixos ou móveis;

- **Deliberativos ou reativos**. Agentes deliberativos possuem simbólos 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.

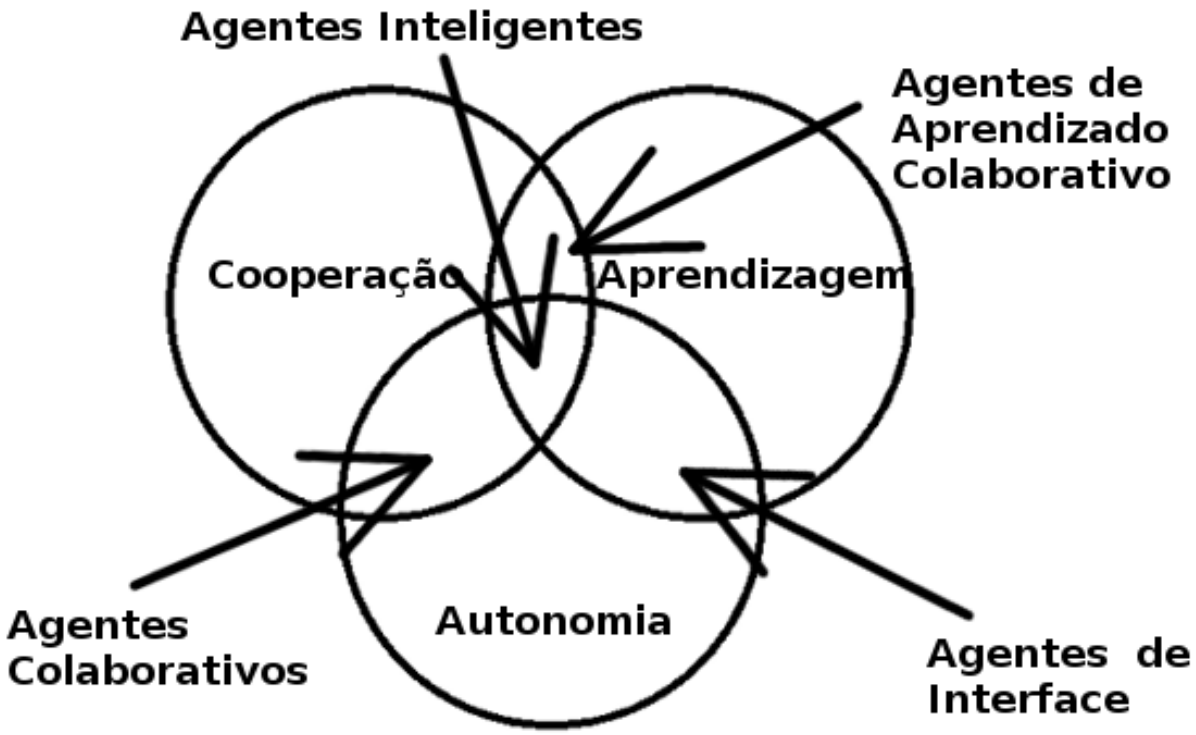

Figura 2.1: Visão parcial da tipologia de um Agente.

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.

Emboras as diferentes definições muita coisas são comuns ou muito parecidas como a presença de características como autonomia, aprendizado, objetivos e comunicatividade.

2.2 Agentes em jogos

Antigamente maioria do processamento do sistema que rodava um jogo(computador, video-game entre outros) era utilizada em sua grande maioria 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. Segundo [van Lent et al., 1999] "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, [Millington and Funge, 2009] mostrou que um número cada vez maior de jogos tem como principal ponto do jogo a Inteligência Artificial. 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 artifical visto em um jogo, com um cerébro baseado em rede neural para cada criatura presente no jogo.

Agentes para jogos tem uma infinidade de própositos. Ao associar esses dois termos é bem comum pensar em oponentes controlados por agentes, porém a utilização de agentes vai muito além disso. 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 agentes para 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 MMORPGs online), segundo os autores essa abordagem tem o pontencial 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 aplicações.

2.3 Agentes inteligente contra jogadores

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

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] nogo Max Payne 2 é introduzido

algo chamado opções dinâmicas de dificuldade. Informações do mundo 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 nos jogos anteriores define qual o nível do jogador, ou seja, quanto mais o jogador evolui mais dificil será seu oponente.

Outra abordagem interessante é a utilizada em Drivatar [Herbrich, 2011] inserida no jogo Forza Motorsport onde o sistema 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* nessa técnica não é 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 [ibm, 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 levado 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 pois temos uma árvore pequena de possibildades e o resultado é vitória, empate ou derrota, ou seja, não é necessário a cada nó da árvore uma complexa avaliação de heurística.

Segundo *Nielsen:1994:HE:189200.189209* 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] dicorrem 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 computação.

As competições de inteligência para Start Craft que ocorrem 2010 avançou significamente 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 que gere uma economia bem complexa e o vencedor é definido pelo jogador que derrotar todas as tropas 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.

Outra competição um pouco diferente é a AI Birds [aib, 2016], nessa competição os agentes são submetidos a jogar Angry Birds, o objetivo desse jogo é você destruir objetos(ganhando assim pontos) com passáros que são arremessados por um estilingue. Dife-

Figura 2.2: BWAPI em funcionamento.

rente do Star Craft nessa competição não há um enfretamento direto entre os agentes, eles são submetidos a enfrentar o mesmo cenário e ganha 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(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 caputrada do navegador e a partir disso é utilizado um algoritmo de reconhecimento de imagem para identificar o que é cada coisa(os quadrados em volta do objeto é o que a API está reconhecendo).

Neste trabalho além da criação de agentes, será explorado a criação de uma API para comunicação com o jogo Pokémon Showdown que é um simulador de batalhas Pokémon. No capítulo 4 será explicado como funciona o sitema de batalhas. A API possibilita batlahas 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 pode fazer um agente.

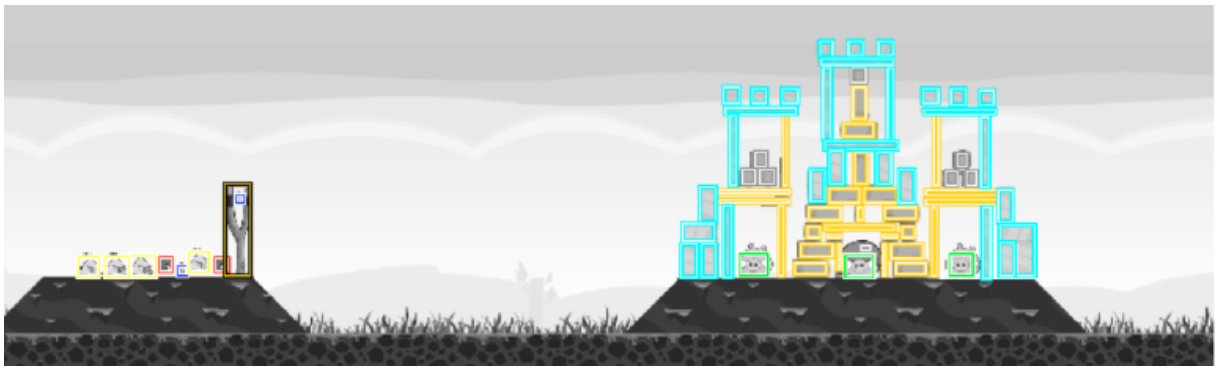

Figura 2.3: Api Angry Birds AI em funcionamento.

Capítulo 3

Inteligência Artificial

Neste capítulo é feito uma revisão das técinicas 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 [**?**].

Segundo [Yannakakis and Togelius, 2015] durante so seminários em Dagstuhl(centro de pesquisa de ciência da computação 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 é utilizado nos jogos e que serão usados neste 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 [aend Edward Powley et al.,
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 árvores de decisões da seguinte forma:

- **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 mais complexas devido a probabildade de um nó acontecer.
- **Miximax** é semelhante ao *expectimax* de apenas um jogador e é usado principalmente em jogos com informações não precisas. Ele usa 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 dar 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 a vez do jogador Min será escolhido o menor valor dos sucessores de p.

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 vitóriosas foi atribuído o valor de 10, em empates 0 e em derrota o valor -10. Em seguida, os pai 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) existem dois nós filhos, um com valor 10 e outro com valor -10, por ser um nó MIN é atribuido para esse nó o menor valor(no caso -10). A linha em azul mostra a melhor jogada no momento para o nó raiz.

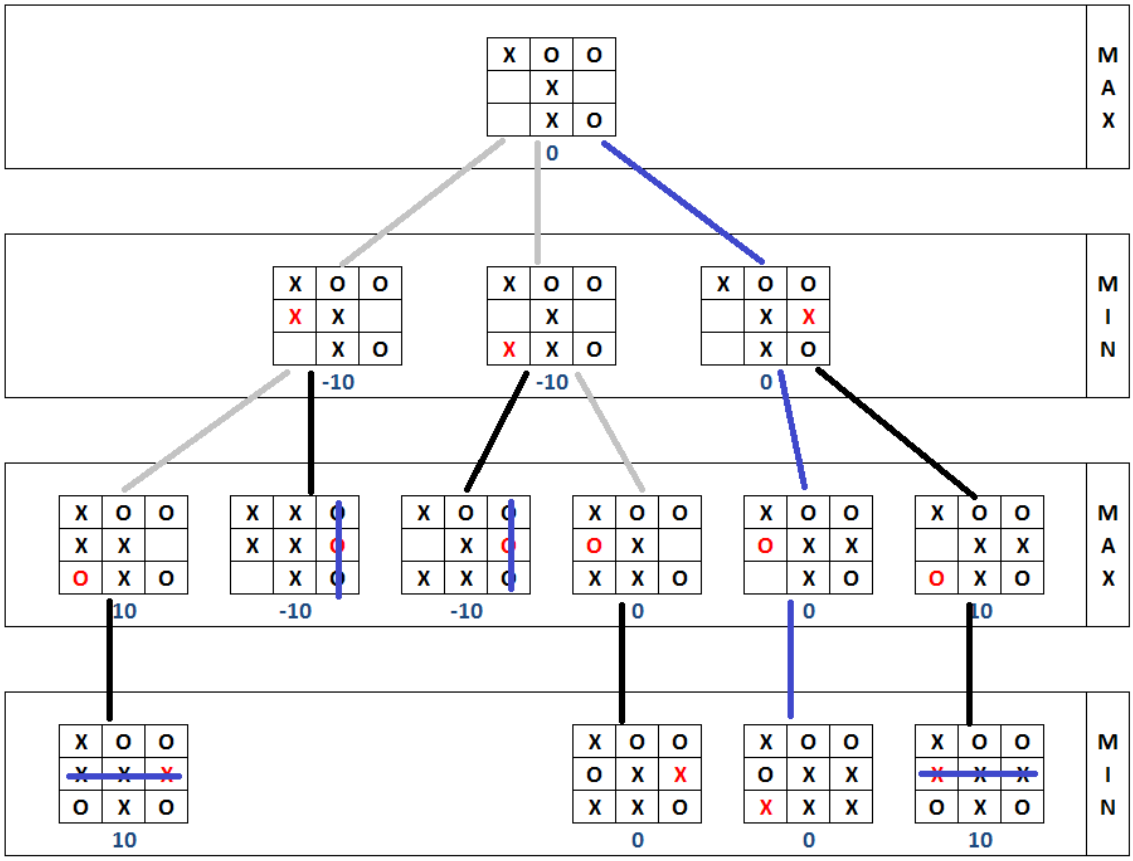

Figura 3.1: *Minimax* aplicado a um cenário de jogo da velha.

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 para o 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 e que duram um grande número de turnos(grande profundidade na árvore). Uma técnica muito utilizada para mitigar esse problema é a chamada poda *alpha-beta*.

Segundo [Knuth and Moore, 1976] essa é tecnica é usada geralmente para aumentar a velocidade de busca sem perder informação. Nessa técnica são ignorado nós e suas sub-árvores de jogadas incapazes de ser melhor que movimentos que já conhecemos. Durante a análise das jogadas são definidas duas váriaveis *alpha* e *beta*. *Alpha* representa o valor máximo que o jogador Max 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 [Campbell and Marsland, 1983] o processo de construnção da MCTS é feita de modo incremental e assimétrico na 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 areas que aindanão tenham sido bem amostradas) e exploração(procura areas que parecem ser promissoras). O nó escolhido é avaliado e todos seus ancetrais 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 respsota 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 pensamenos 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 bem 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.

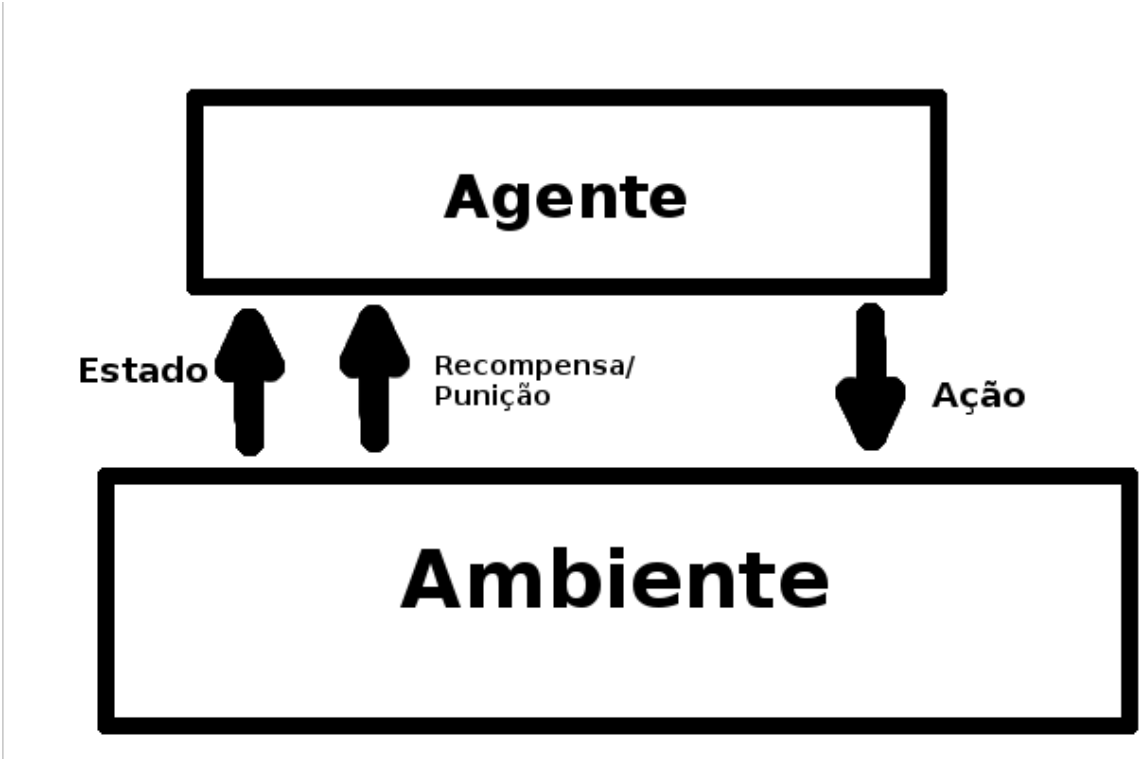

Figura 3.2: Esquema padrão do aprendizado por reforço.

Existe uma série de alterações nessa abordagem para abrangir uma maior gama de problemas. Uma variação bem comum é a situação chamada de recompensa atrasada. 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 qe escolhe. Com isso surge a dúvida qual das estratégias produz o mais efetivo aprendizado (para colher novas informações). Explorar todos as possíveis ações, ou explorar estados e acões já conhecidos com alta recompensa (para maximizar a sua recompensa culmulativa).

- **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 enxegar o ambiente que está atrás dele.

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

Uma das técnicas que resolve esse problema 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 o 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 propridade de Markov, que diz que a transição de estados tem que ser independente de qualquer informação de estados e ações do 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 utilizado 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, idenpendente de como o agente se comporta enquanto os dados estão sendo coletados (desde que todas ações/estados sejão 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 equação 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 pode ser qualquer contstante entre $0 \leq \gamma \leq 1$. Se γ é proximo de zero, o agente tenderá a considerar mais recompensas imediatas, se for próximo de um o agente considerará mais as futuras recompensas,
- **s'**. O próximo estado.
- **a***. Todas as ações do próximo estado.

O treinamento constite 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 um excelentes ambientes de testes para pesquisa de neuroevolução.

3.4.1 Redes Neurais

Segundo [Koehn, 1994] 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) tem 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 pode ser comparado 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(pelo menos um) nó 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. Esse é o vetor onde os valores serão ajustados até que a saída fique correta,
- Bias b é um valor de polarização permite mudar a função de ativação para a esquerda ou para a direita, ou seja, essa é uma variável que também é ajustada pelo modelo e ela permite melhor espalhar os resultados que a função de ativação gera.

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 entradas, todas saídas estão corretas o vetor W e o bias é atualizado. Segundo [Estebon,] ajustando as pesos das conexões entre ascamadas, a saida do *perceptron* pode ser treinada para corresponder uma determinada saída. 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 dos valores desejados.

A equação de treinamento pode ser escrita da seguinte forma:

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

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 sejão ajustados mais suavementes.
- t_j Resultado esperado.
- x_j Resultado atual.
- a_i Valor de correção. Se o peso não precisa ser ajustado o valor será 0, caso contrário será 1.

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 cromosomo de modo que ele representa uma solução para o problema. Para usar o modo convencional do algoritmo genético com cromosomos de tamanhos fixados é preciso:

- 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 cromosomo depende de quão bem o cromosomo resolve o problema.
- Determinar os parâmetros e variáveis para controlar o algoritmo.
- Determinar um modo de mostrar o resutlado 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 cromosomos(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 cromosomos através de alguma operação de cruzamento (*crossover*); mutar algumas características de um cromossomo randomicamente.
- 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 excelentes 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 da herança genética.

Existe também outros modos de aplicar algoritmos evoluitivos 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. A performance ou *fitness* da rede é então guardado, uma vez que os valores de aptidão para os genótipos(indivíduos) na população atual são determinados, uma nova população é gerado 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

Batalhas Pokémon

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 categorizado a evolução no ranquemanto do jogo de acordo com o aumento de profundidade da árvore de decisão.

- **Agente 1 com neuroevolução**. O primeiro agente persiste em uma rede neural onde os pesos de sua rede serão ajustados por um algoritmo evolutivo.
- **Agente 2 com aprendizado por reforço**. Esse agente irá se ajustar através de estimulos positivos e negativos que regularão sua decisão dentro do jogo através do algoritmo de aprendizado por reforço.
- **Agente 3 baseado em grafo de decisão**. Utilizará uma árvore de decisão 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(grafo). Várias versões do agente serão criadas, cada versão será cablibrada com uma profundidade diferente em sua árvore. Cada versão do agente será submetido a uma série da batalhas que só terá fim quando seu ranqueamento se estabilizar.

Os agentes 1 e 2 terão comportamentos semelhantes em como serão treinados. Ambos agentes terão duas versões, cada versão fará um treinamento diferente:

- **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 apenas contra o agente 3, sempre que o agente que está treinando conseguir um grande números 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 dele(quantidade de pontos de vida, afetado por alguem efeito negativo entre outros).

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

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."[aib, 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 Pokemon 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.

- **Escever 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.
- **Escever 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									
AIB	X	X	X														
API							X	X	X								
EDQ									X	X							
DTA										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

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

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

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

[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*.

[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.

[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.

[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,] Estebon, M. D. 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.

[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–.

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.