UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Gabriel Gomes Pereira

DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA O JOGO 7 WONDERS

Santa Maria, RS
2021

Gabriel Gomes Pereira

DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA O JOGO 7 WONDERS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação, Área de Concentração em ciências exatas e da terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Sistemas de Informação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. Joaquim Vinicius Carvalho Assunção

Santa Maria, RS
2021

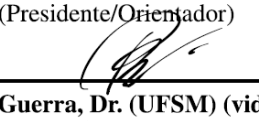
Gabriel Gomes Pereira

DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA O JOGO 7 WONDERS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação, Área de Concentração em ciências exatas e da terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Sistemas de Informação**.

Aprovado em 12 de fevereiro de 2021:

Joaquim Vinicius Carvalho Assunção, Dr. (UFSM)
(Presidente/Orientador)



Rodrigo da Silva Guerra, Dr. (UFSM) (videoconferência)

Luís Alvaro de Lima Silva, Dr. (UFSM) (videoconferência)

Santa Maria, RS
2021

DEDICATÓRIA

Dedico este trabalho ao meu avô Ugo (in memorian) e a minha avó Lacy, que me ensinaram a ter fé, persistência e fazem parte minhas maiores motivações nessa caminhada, a minha avó Eva (in memorian) que foi um exemplo de força e aos meus pais José de Alencar e Rozeli que sempre me deram apoio e batalharam muito para que eu chegasse até aqui.

AGRADECIMENTOS

Agradeço ao professor Joaquim Vinicius Carvalho Assunção por ter me orientado com tanta dedicação e paciência. Também sou grato por todos os ensinamentos e por ter acreditado no meu trabalho.

Também deixo meu agradecimento a todo o grupo de pesquisa DMAG, a dedicação e o esforço de todos durante estes 2 anos criou uma base para que fosse possível desenvolver este trabalho.

RESUMO

DESENVOLVIMENTO DE AGENTES INTELIGENTES PARA O JOGO 7 WONDERS

AUTOR: Gabriel Gomes Pereira
ORIENTADOR: Joaquim Vinicius Carvalho Assunção

A construção de agentes para jogos utilizando técnicas e algoritmos de inteligência artificial é uma prática comum. Nesse contexto, esse trabalho teve por objetivo desenvolver agentes inteligentes para o jogo *7 Wonders*. Para isso, foram desenvolvidos três agentes, o primeiro utiliza regras que se baseiam em estratégias obtidas em um trabalho prévio e não possui qualquer tipo de aprendizado, o segundo agente implementado utiliza redes neurais treinadas de forma supervisionada e o terceiro agente foi desenvolvido utilizando uma técnica de aprendizado por reforço profundo chamada de *Deep Q-Learning*. O agente baseado em redes neurais e o agente baseado em *Deep Q-Learning* foram capazes de aprender estratégias utilizadas pelos melhores jogadores de uma plataforma chamada *Board Game Arena* (BGA). Nos testes contra um agente que faz apenas jogadas aleatórias, todos os três agentes obtiveram uma pontuação média satisfatória e próxima das pontuações dos melhores jogadores do BGA.

Palavras-chave: Redes Neurais. Inteligência Artificial. Aprendizado de máquina. Jogos. *Deep Q-Learning*.

ABSTRACT

DEVELOPMENT OF INTELLIGENT AGENTS FOR *7 WONDERS* GAME

AUTHOR: Gabriel Gomes Pereira
ADVISOR: Joaquim Vinicius Carvalho Assunção

The construction of agents for games using artificial intelligence techniques and algorithms is a common practice. In this context, this work aimed to develop intelligent agents for the game 7 Wonders. For this purpose, three agents were developed, the first uses rules that are based on strategies obtained in previous work and do not have any type of learning, the second agent implemented uses supervised neural networks and the third agent was developed using a deep reinforcement learning technique called *Deep Q-Learning*. The agent based on neural networks and the agent based on Deep Q-Learning were able to learn strategies used by the best players on a platform called Board Game Arena (BGA). In tests against an agent who makes only random plays, all three agents obtained a satisfactory average score and close to the scores of the best BGA players.

Keywords: Neural Networks. Artificial Intelligence. Machine Learning. Games. Deep Q-Learning.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

JSON	*Javascript Object Notation*
BGA	*Board Game Arena*
CSV	Valores Separados por Vírgula
IA	Inteligência Artificial
RNA	Rede Neural Artificial
MLP	*Multilayer Perceptron*
ReLu	*Rectified Linear Unit*
Tanh	Tangente Hiperbólica
ID	Identificador Único
MSE	*Mean Square Error*

SUMÁRIO

1 INTRODUÇÃO

O estudo e aplicação de técnicas e algoritmos de inteligência artificial (IA) na construção de agentes para jogos, sejam digitais ou analógicos, é uma prática frequente. A natureza complexa, competitiva e às vezes imprevisível dos jogos os tornam objetos de estudo interessantes para a IA (RUSSELL; NORVIG, 2013). Existem trabalhos bastante conhecidos que construíram agentes inteligentes para jogos populares como Xadrez (CAMPBELL; JR; HSU, 2002), Damas (SCHAEFFER et al., 2007), *Starcraft 2* (VINYALS et al., 2017) e *Go* (SILVER et al., 2017), além disso os dois últimos trabalhos mencionados utilizam técnicas de aprendizagem de máquina.

O campo da IA que vem crescendo cada vez mais nas últimas décadas é o Aprendizado de Máquina. As Redes Neurais Artificiais (RNAs) fazem parte da ampla gama de técnicas de aprendizado de máquina disponíveis. Entre os exemplos de aplicação de modelos de RNAs estão: classificação de imagens, reconhecimento de fala e processamento de linguagem natural.

As RNAs são aplicadas a diversas áreas, incluindo também os jogos. Dessa forma, modelos podem ser construídos para aprender as regras e estratégias competitivas para um determinado jogo. Nesse sentido podemos citar o sucesso obtido pela inteligência artificial nomeada como Alpha Go (SILVER et al., 2017), após derrotar o campeão mundial Lee Sedol.

Além disso, sabe-se que a construção de modelos capazes de aprender a jogar determinados jogos não é uma tarefa trivial, pois existem jogos que em termos de representação computacional possuem uma grande quantidade de estados, ações e estratégias possíveis. Esse tipo de ambiente necessita que sejam construídos modelos complexos e que envolvem um grande custo computacional. Este cenário aplica-se ao jogo de tabuleiro 7 Wonders.

Para que um bom modelo seja construído, é necessário ter o conhecimento sobre as ações que fornecem as melhores chances de vitória em um jogo. Nesse contexto, o trabalho de Assunção et al. (2019), utilizou regras de associação para obter o conhecimento de estratégias, que aumentam as chances vitória, relacionadas ao jogo *7 Wonders*.

Complementar ao trabalho de Assunção et al. (2019), foi desenvolvida uma versão digital do jogo *7 Wonders* por Jardim et al. (2020)[1]. Essa implementação fornece um ambiente que visa facilitar o teste de agentes inteligentes. Sendo assim, com a disponibilidade de uma versão digital do jogo e o conhecimento de boas estratégias, temos um ambiente propício para a construção de modelos de Aprendizado de Máquina que sejam capazes de jogar *7 Wonders*.

O *7 Wonders* é um jogo destinado a grupos de 3 a 7 jogadores, constituído por 7 tabuleiros que representam as 7 maravilhas do mundo antigo e 148 cartas divididas em 7 tipos, onde o objetivo é obter o maior número de pontos de vitória ao fim de uma partida. Em termos de representação computacional, o jogo possui uma grande quantidade de possíveis estados e envolve um ambiente de tomada de decisões que deve considerar diversas variáveis durante uma

[1] https://github.com/dmag-ufsm/7Wonders

partida. As regras e detalhes são esclarecidos na Seção 2.1.

O objetivo deste trabalho foi construir agentes inteligentes capazes de jogar a versão de 3 jogadores do jogo *7 Wonders*, também foi realizada uma comparação dos mesmos. Logo, esta monografia descreve um trabalho inicial, onde foram desenvolvidos 3 agentes que utilizam:

1. regras fixas;
2. uma rede neural treinada de forma supervisionada e;
3. um agente que utiliza uma técnica de aprendizado por reforço profundo chamada de *Deep Q-Learning*.

Este trabalho é um dos primeiros a explorar o desenvolvimento de agentes inteligentes para o jogo *7 Wonders*, que é um dos jogos de tabuleiro mais populares do mundo e também o mais premiado. Foram dados os primeiros passos, utilizando diferentes abordagens, para a construção de agentes competitivos.

A escrita foi organizada da seguinte maneira, no Capítulo 2 são apresentados os conceitos necessários para o desenvolvimento do trabalho. Nos Capítulos 3 e 4, são esclarecidos, respectivamente, os métodos utilizados para a construção dos agentes e realizada discussão dos resultados. E por fim no Capítulo 5 é realizada a conclusão.

2 REFERENCIAL TEÓRICO

Este capítulo visa apresentar e esclarecer os conceitos necessários para o desenvolvimento desse trabalho. Dessa forma são apresentadas as regras do jogo, conceitos envolvendo as técnicas de aprendizagem de máquina utilizadas neste trabalho, e trabalhos relacionados.

2.1 SOBRE O 7 WONDERS

0 7 Wonders é um jogo de tabuleiro criado por Antoine Bauza, destinado a grupos de 3 a 7 jogadores. O jogo possui 7 tabuleiros, que representam as 7 maravilhas do mundo antigo. Uma partida é divida em 3 eras, cada uma contendo um baralho diferente. O objetivo está em obter a maior quantidade de pontos de vitória até o final de uma partida, por meio da construção de estruturas, que são representadas por cartas, e estágios da maravilha. A Figura 2.1 mostra os elementos do jogo.

Figura 2.1 – Visão geral dos elementos do jogo 7 Wonders.

Fonte: Adaptado de (BAUZA, 2010, p. 2)

Cada tabuleiro, aqui simplesmente chamado de maravilha, fornece um recurso que pode ser uma matéria prima ou produto manufaturado, e possui estágios que ao serem completados recompensam o jogador com mais recursos, moedas, pontos de vitória ou outras vantagens ao longo de uma partida. A Figura 2.2 mostra uma maravilha com seus lados A e B.

Além dos tabuleiros, as cartas compõem outro elemento principal do jogo, elas representam estruturas, ilustradas na Figura 2.3, que estão divididas em 7 tipos que são:

- As cartas de matéria prima fornecem recursos como argila, pedra, minério e madeira, que

[1]Disponível em <https://boardgamearena.com/> Acesso em 19 de novembro de 2020.

Figura 2.2 – Maravilha representando *Os Jardins Suspensos da Babilônia*.

Fonte: Captura retirada de *Board Game Arena*[1]

muitas vezes são necessários para a construção de estágios da maravilha e dos demais tipos de estruturas com exceção dos produtos manufaturados.

- Os produtos manufaturados desempenham função semelhante às cartas de matéria prima e abastecem o jogador com recursos como tecido, vidro e papiro.
- As estruturas civis, quando construídas, tem o efeito de dar pontos de vitória ao jogador.
- As cartas que representam estruturas comerciais, podem fornecer moedas e vantagens no comércio que são úteis para a compra de recursos.
- As estruturas militares, fornecem escudos que são contabilizados em uma etapa do jogo, que ocorre ao final de cada era, chamada de "Conflitos Militares".
- As estruturas científicas possuem mecânicas de pontuação baseadas nas combinações de símbolos contidos nelas. Para cada conjunto de 3 cartas com símbolos distintos, o jogador recebe 7 pontos de vitória, além do mais é somado a isso o quadrado da quantidade de símbolos iguais.
- As guildas fornecem pontos de vitória utilizando critérios baseados no contexto da partida, geralmente com base nas cartas dos oponentes.

Quando uma partida começa, cada jogador recebe 7 cartas. Após, uma carta deve ser escolhida para uma ação e revelada somente ao final do turno, quando todos tiverem realizado uma

[2]Disponível em <http://www.asmodee.es/juegos/coleccion/7_wonders> Acesso em 19 de novembro de 2020.

Figura 2.3 – Cartas representando diferentes tipos de estruturas.

Fonte: *Asmodee*[2]

escolha. O jogador pode tomar 3 ações com a carta selecionada: construir estrutura, construir estágio da maravilha ou descartar a carta.

Para qualquer ação relacionada a construir, o jogador deve possuir os recursos requisitados pela carta. Caso contrário, a estrutura deve ser descartada e o jogador recebe 3 moedas. Após realizada uma ação, as cartas restantes devem ser passadas ao jogador vizinho. Isso ocorre até o sexto turno, que marca o fim de uma era na partida. A carta restante é descartada e logo após ocorre a etapa de conflitos militares.

Durante essa etapa, cada jogador confronta todos os vizinhos e para cada vitória, dependendo da era em que se encontra, recebe uma quantidade de pontos que são somados à pontuação total. Para isso, são comparadas as quantidades de escudos, sendo o ganhador o que possuir mais em relação ao vizinho. Além disso, ao perder um conflito é descontado 1 ponto.

Por fim, uma partida tem a duração de 3 eras, contendo 6 turnos cada. Além disso, como vimos, o jogo envolve tomadas de decisão em um ambiente com informação imperfeita, ou seja não temos acesso a todas as informações da partida. Nesse contexto, a aplicação de algoritmos de aprendizado de máquina, como modelos de redes neurais e técnicas de aprendizagem por reforço profundo se tornam um desafio interessante.

2.2 APRENDIZADO DE MÁQUINA

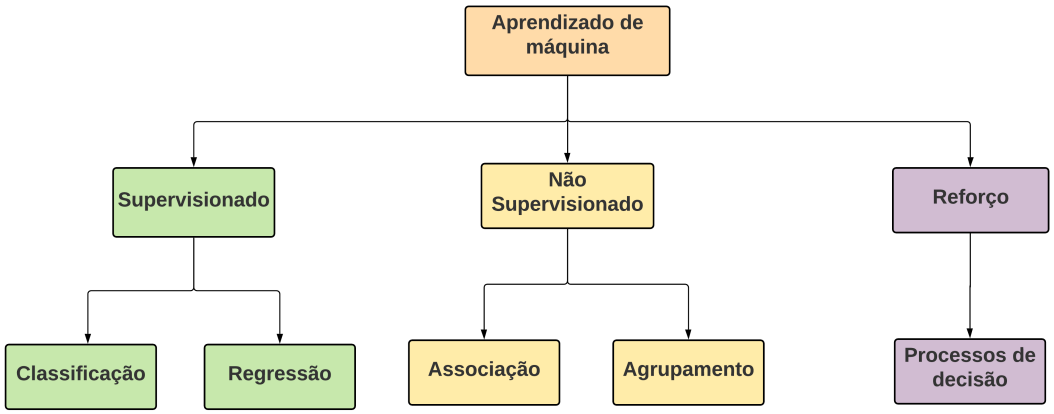
Segundo Faceli et al. (2011) aprendizado de máquina refere-se a um conjunto de técnicas que tem por objetivo construir modelos capazes de aprender com experiências passadas. Para isso computadores são programados para encontrar padrões em conjuntos de dados de forma a

obter conclusões genéricas a respeito do problema que se pretende resolver.

Faceli et al. (2011) divide tarefas de aprendizado em dois grupos: preditivas e descritivas. As tarefas preditivas, precisam ser treinadas com exemplos de dados, com atributos de entrada e saída, para que possam ser gerados modelos capazes de prever rótulos ou valores de novas entradas. Quando o propósito do algoritmo é encontrar relações entre os atributos de um conjunto de dados, sem se preocupar com as saídas, ele faz parte de uma tarefa de descrição.

Além do mais, as tarefas discutidas anteriormente definem dois paradigmas: o aprendizado supervisionado (predição) e não supervisionado (descrição). Ainda vale mencionar um terceiro paradigma chamado de aprendizado por reforço, onde a aprendizagem ocorre por meio de punições ou recompensas. A Figura 2.4 mostra a divisão entre os paradigmas de aprendizado de máquina, e de forma geral os tipos de algoritmos que cada um abrange.

Figura 2.4 – Tipos de aprendizado de máquina.



Fonte: Criado pelo próprio autor.

Quanto à aplicação de aprendizado de máquina em jogos, agentes podem aprender a jogar um determinado jogo utilizando aprendizado supervisionado. Conjuntos de dados contendo históricos de jogadas de jogadores humanos podem ser utilizados para treinar algum aproximador de função (por exemplo, redes neurais), com o intuito de fazê-lo jogar como um humano. O conjunto de dados deve conter as tuplas <Estado, Ação>, para que o modelo aprenda a reproduzir as ações realizadas por humanos em determinados estados (YANNAKAKIS; TOGELIUS, 2018).

Além disso, a aprendizagem por reforço é frequentemente utilizada em ambientes de jogos. Um agente pode aprender a seguir determinadas estratégias de um jogo, por meio de tentativa e erro. Para isso são fornecidas recompensas ao longo do tempo, que podem ser quantidades de pontos, e punições (um valor negativo). Dessa forma, algoritmos de aprendizado por reforço buscam otimizar o comportamento do agente para maximizar a recompensa acumulada (YANNAKAKIS; TOGELIUS, 2018).

2.3 REDES NEURAIS

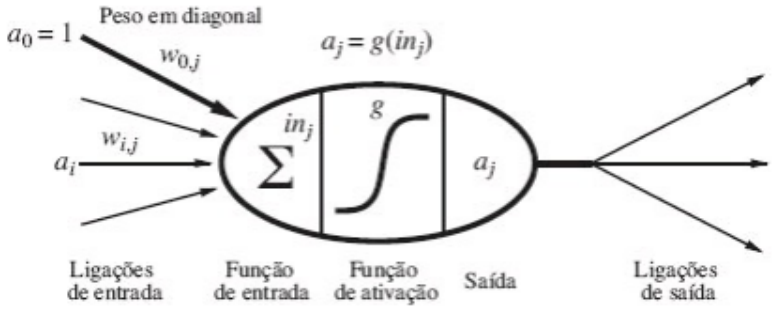
As Redes Neurais Artificiais (RNAs) são sistemas compostos por unidades de processamento conhecidas como neurônios que são responsáveis por calcular funções matemáticas. Essas unidades estão dispostas em uma ou mais camadas, e são interligadas por conexões normalmente unidirecionais. O conhecimento adquirido por uma rede é armazenado na forma de pesos que estão associados às conexões. Esse modo de funcionamento é inspirado nas redes neurais biológicas (BRAGA; LUDERMIR; CARVALHO, 2000).

Um dos estudos considerados pioneiros na área, foi produzido por McCulloch e Pitts (1943) que propuseram um modelo matemático para representação de um neurônio artificial (FACELI et al., 2011). Mais tarde, durante a década de 1960, este modelo foi aprimorado por Frank Rosenblatt, que propôs um modelo com algoritmo de aprendizagem chamado de *perceptron* (ALPAYDIN, 2014).

2.3.1 Perceptron

O modelo de neurônio artificial proposto por McCulloch e Pitts (1943), mostrado na Figura 2.5, serve como referência para o *perceptron* construído por Rosenblatt (HAYKIN et al., 2009). Uma rede neural do tipo *perceptron* possui todas as entradas, que são valores representando informações relacionadas ao problema que se pretende resolver, ligadas a apenas um neurônio de saída. Além disso, a ligação entre cada entrada e saída possui um peso responsável por armazenar o conhecimento adquirido pela rede, que pode ser inicialmente definido com base em algum critério ou de forma aleatória.

Figura 2.5 – Modelo de neurônio proposto por McCulloch e Pitts(1943).



Fonte: (RUSSELL; NORVIG, 2013, p. 635)

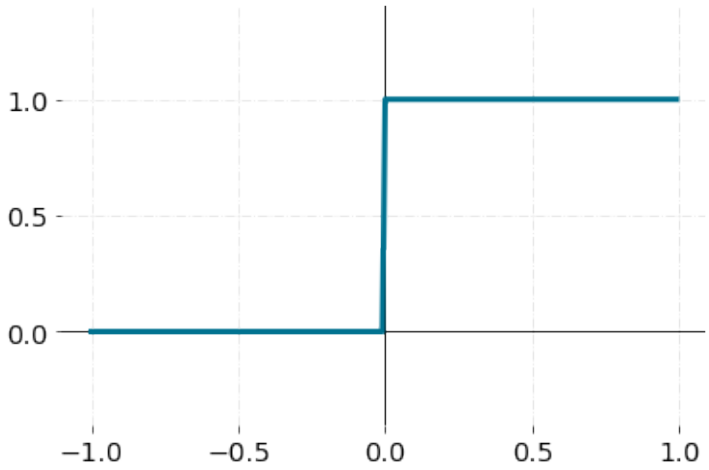
Sendo assim, um neurônio artificial recebe entradas que podem ser representadas como um vetor $a = [a_1, a_2, a_3, ..., a_n]$ e que estão associadas a pesos de ligação $w = [w_1, w_2, w_3, ..., w_n]$ e um bias a_0 associado a um peso w_0 (RUSSELL; NORVIG, 2013). Dessa forma, a função de entrada de um neurônio é definida pela soma ponderada das entradas, representada na equação

2.1.

$$\sum_{i=0}^{n} a_i w_i \tag{2.1}$$

Depois de passar pela função de entrada, o resultado passa por uma função de ativação f cujo resultado define a ativação de uma determinada saída. No modelo de McCulloch e Pitts (1943), é utilizada a função degrau, mostrada na Figura 2.6, que define uma saída com resultado 0 para valores de entrada negativos e 1 para valores positivos (BRAGA; LUDERMIR; CARVALHO, 2000).

Figura 2.6 – Função Degrau.



Fonte: Criado pelo próprio autor

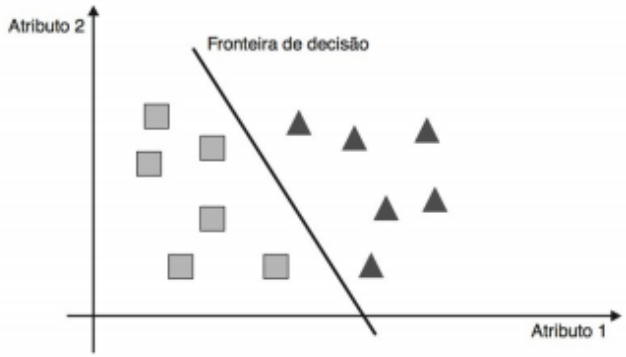
A aprendizagem em redes perceptron ocorre de maneira supervisionada e iterativa, de forma que a cada execução um algoritmo realiza a correção de erros, comparando a saída esperada com a que foi fornecida pela rede (FACELI et al., 2011). Esse modelo de aprendizado é utilizado em tarefas de classificação binária.

No entanto, o modelo limita-se a problemas que são linearmente separáveis (BRAGA; LUDERMIR; CARVALHO, 2000). Ou seja, perceptrons só funcionam para problemas de classificação onde podemos traçar uma reta no plano cartesiano bidimensional para obter duas classes distintas, como mostrado na Figura 2.7. Já as redes perceptron multicamadas, do inglês *Multilayer Perceptron* (MLP), por possibilitarem a utilização de funções de ativação não lineares, são livres dessa limitação.

2.3.2 Perceptron Multicamadas

As redes *perceptron* multicamadas são semelhantes às redes *perceptron*, no entanto, possuem uma ou mais camadas intermediárias com um número indeterminado de neurônios. Este

Figura 2.7 – Objetos linearmente separáveis



Fonte: (FACELI et al., 2011, p. 115)

tipo de RNA é capaz de resolver problemas de classificação e regressão não-lineares (ALPAYDIN, 2014). Para isso, essas RNAs devem ser projetadas utilizando funções de ativação de natureza não-linear em suas camadas.

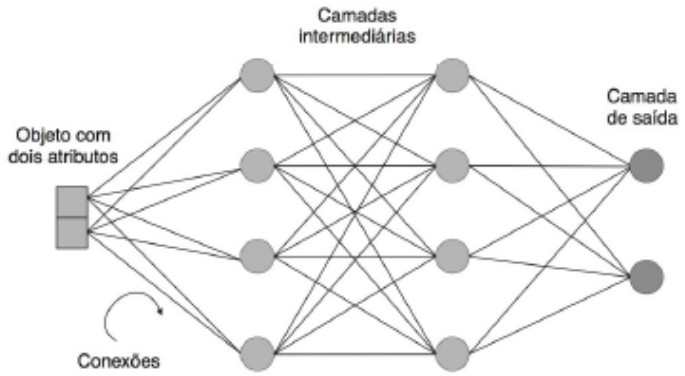
A escolha de uma função de ativação pode ter impacto significativo na capacidade e velocidade de aprendizado de uma RNA. Entre as funções de ativação mais utilizadas em arquiteturas de RNAs estão Sigmóide, *Rectfied Linear Unit* (ReLu), *Leaky ReLu*, *Softmax* e Tangente Hiperbólica (Tahn).

Faceli et al. (2011) diz que a topologia de uma RNA é definida pelo número e grau de conexão entre os neurônios existentes em cada camada, também deve ser considerada a existência ou não de conexões de retroalimentação. Além disso, de acordo com o grau de conectividade entre os neurônios, uma rede pode ser totalmente conectada, parcialmente conectada ou localmente conectada. Entre os tipos mais conhecidos e utilizados estão as redes totalmente conectadas.

As RNAs totalmente conectadas são organizadas de forma que os neurônios de uma camada possuem conexões com todos os neurônios da camada seguinte, como mostrado na Figura 2.8. Essas conexões recebem pesos de ligação, que são responsáveis por armazenar o conhecimento obtido pelo modelo. Esta é uma das arquiteturas de RNAs mais utilizadas na prática e além disso são chamadas de redes *feedforward* (FACELI et al., 2011).

Nas redes *feedforward*, o fluxo das informações geralmente ocorre de tal forma que a partir da entrada, as saídas das funções de ativação do conjunto de neurônios de uma camada são enviadas para a seguinte até chegarem na saída da rede. Dessa forma os neurônios de entrada geram um padrão de ativação que produz uma resposta na camada de saída (FACELI et al., 2011; HAYKIN et al., 2009).

Figura 2.8 – Exemplo de rede neural multicamadas completamente conectada.



Fonte: (FACELI et al., 2011, p. 111)

2.3.3 Fase de treino

A fase de treino de uma RNA *Multilayer Perceptron* inicia com a seleção do conjunto de dados que será utilizado para treinar a rede. Esse conjunto contém informações correspondentes às entradas (características) e saídas esperadas para a RNA. Além disso os dados são separados em treino e teste.

Os dados de treino são apresentados durante o treinamento para que a RNA aprenda as relações entre as entradas e saídas esperadas. E os dados de teste, que são amostras que ainda não foram mostradas à rede, são apresentados após o treino para testar a capacidade de generalização da rede.

Ao iniciar o treinamento, os dados de entrada são colocados na primeira camada da rede e propagados camada a camada até chegarem na saída, essa etapa é denominada como *forward*. Ao final da etapa de *forward* as saídas produzidas pela RNA são comparadas com os dados relacionados às saídas esperadas e é realizado o cálculo do erro (BRAGA; LUDERMIR; CARVALHO, 2000).

O erro é calculado por uma função de custo, entre as mais conhecidas estão a *cross-entropy*, recomendada para tarefas de classificação e o *Mean Square Error* (MSE), indicado para aplicações que utilizam regressão (GOODFELLOW; BENGIO; COURVILLE, 2016). Após isso, o erro flui da camada de saída para a entrada, essa etapa é chamada de *backward*.

Durante a etapa *backward*, os pesos das conexões entre os neurônios de cada camada são ajustados com o objetivo de minimizar o erro, ou seja, aproximar a saída da rede a resposta esperada. Geralmente o treino termina quando a RNA atinge uma taxa de erro aceitável.

Além disso, ao fim do treinamento são observados outros 2 parâmetros relacionados à precisão de uma RNA. A acurácia de treino está relacionada ao percentual de acertos que uma rede consegue obter quando são apresentados os dados de treino. E a acurácia de teste, se refere ao percentual de acertos quando são apresentadas as entradas do conjunto de teste.

Ao observar a acurácia de treino e teste, pode ser constatado um problema chamado

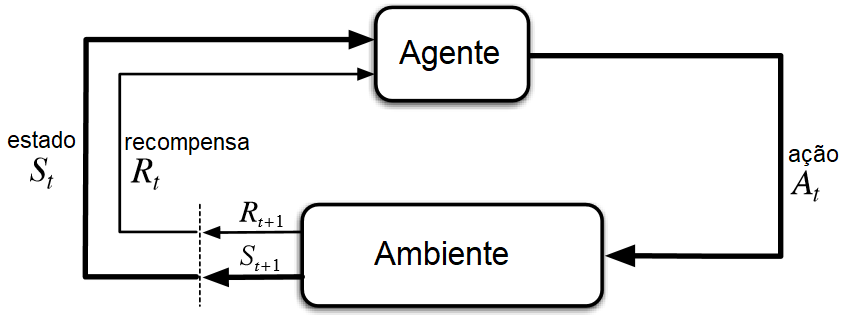
overfitting, que ocorre quando a rede neural se ajusta muito bem aos dados de treinamento, mas não consegue fornecer as saídas corretas para grande parte dos dados de teste. Nesse caso, a acurácia de treino acaba se mostrando maior que a de teste. Além disso, se diz que a rede neural não possui uma boa capacidade de generalização, ou seja, não mostra um bom desempenho ao prever novos dados.

2.4 DEEP Q-LEARNING

O *Deep Q-Learning* proposto pelos pesquisadores da *Deep Mind Technologies* é uma técnica que une *Deep Learning* e aprendizagem por reforço. Esta é uma solução utilizada na construção de agentes inteligentes para jogos que possuem um grande número de estados e ações (DATA SCIENCE ACADEMY, 2020). Para entendermos o seu funcionamento precisamos inicialmente entender o que é aprendizado por reforço e *Q-Learning*.

Ao contrário da aprendizagem supervisionada, onde um modelo é treinado com base num conjunto de entradas e saídas desejadas. No aprendizado por reforço os modelos são treinados para atingir um determinado objetivo enquanto exploram o ambiente, por meio de punições e recompensas. Além disso, os algoritmos de aprendizagem por reforço possuem 5 componentes principais: o agente, as ações, as recompensas, o estado e o ambiente (DATA SCIENCE ACADEMY, 2020). A Figura 2.9, mostra a estrutura do processo de aprendizagem por reforço.

Figura 2.9 – Estrutura do processo de aprendizagem por reforço.



Fonte: Adaptado de (SUTTON; BARTO, 2018, p. 48)

A exploração do ambiente ocorre de modo que o agente começa testando ações totalmente aleatórias e conforme recebe um *feedback*, que são as punições e recompensas, aprende a empregar táticas cada vez melhores para chegar no objetivo. A ideia deste método é fazer com que sejam escolhidas as melhores ações, assim maximizando a recompensa recebida.

Um dos algoritmos mais utilizados na aprendizagem por reforço é o *Q-Learning*, esse algoritmo mede o quão valiosa será uma determinada ação ou jogada em um determinado estado. Desse modo o agente escolhe a ação que possuir o maior valor e que terá como retorno as

melhores recompensas futuras.

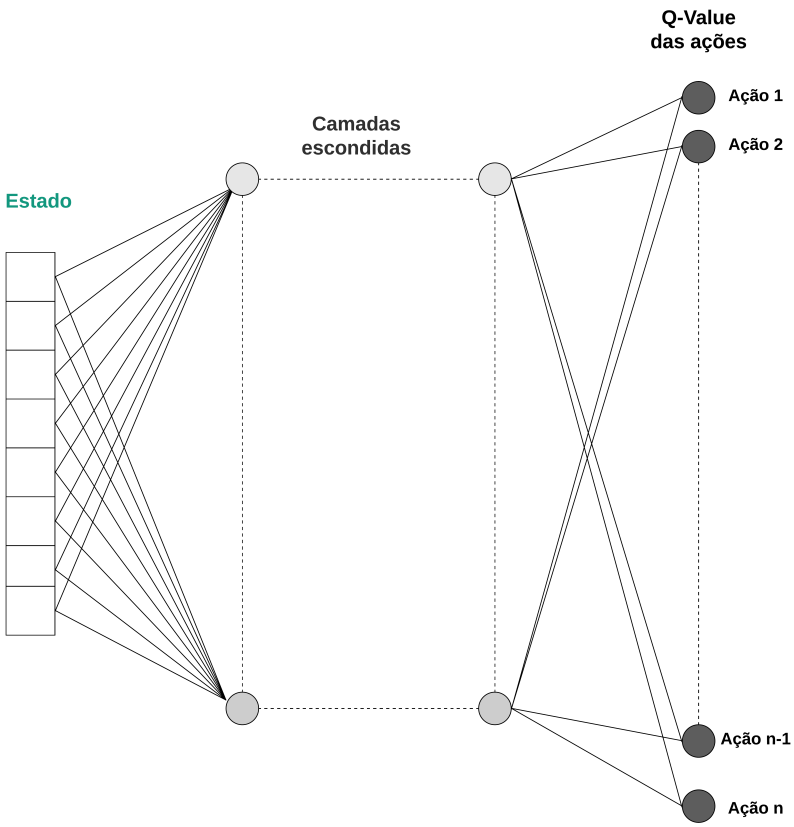
O *Q-Learning* faz uso de uma tabela que mapeia as relações entre os valores Q, que são a qualidade ou valor de uma ação a, e os estados s do ambiente. Os valores dessa tabela são atualizados utilizando a equação de Bellman 2.2.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a) \tag{2.2}$$

A equação de Bellman diz que o valor de $Q(s,a)$, que corresponde à qualidade de tomar uma ação a no estado s, é dado pela soma da recompensa imediata recebida, representada por $r(s,a)$, com o maior valor que pode ser obtido no estado seguinte s', ao executar a ação a. O fator de desconto, que é representado por γ (gama), define o grau de importância das recompensas futuras.

Quando o número de estados e ações possíveis em um jogo é muito grande, a construção da tabela para os valores de Q possui um custo computacional muito alto. A solução para isso é a utilização de *Deep Q-Learning*. A principal ideia dessa técnica é substituir a tabela por uma RNA que recebe como entrada um estado e produz na sua saída valores relacionados a cada uma das ações possíveis como ilustrado na Figura 2.10.

Figura 2.10 – Estrutura de uma rede para *Deep Q-Learning*



Fonte: Criado pelo próprio autor

No *Deep Q-Learning* uma RNA é utilizada para estimar os valores de Q, em outras

palavras, o papel da rede é fornecer os valores de cada ação para um determinado estado. Para que a RNA aprenda estes valores, o cálculo do erro na saída da rede é adaptado para utilizar a equação de Bellman e acaba sendo geralmente realizado utilizando a função *Mean Square Error* (MSE) (DATA SCIENCE ACADEMY, 2020).

2.5 TRABALHOS RELACIONADOS

Mnih et al. (2013) propôs uma variante do algoritmo *Q-Learning* conhecida como *Deep Q-Learning*. Neste trabalho os pixels que representam o estado do jogo são enviados para a entrada de uma Rede Neural Convolucional que estima o valor para todas as ações disponíveis. Dessa forma, o agente escolhe as ações de maior valor com o intuito de maximizar as recompensas recebidas. Esse método foi testado em sete jogos de *Atari 2600* e se mostrou superior à maioria das abordagens em 6 dos jogos. Além disso, a inteligência artificial conseguiu superar especialistas humanos em 3 dos jogos testados.

Robilliard, Fonlupt e Teytaud (2014) utilizaram o algoritmo *Monte-Carlo Tree Search* (MCTS) no desenvolvimento de um agente inteligente para jogar *7 Wonders*. A árvore construída armazena todas as ações possíveis para partidas com 3 jogadores. O objetivo do trabalho foi testar a eficiência do algoritmo utilizado no agente implementado para jogar *7 Wonders*. Para isso, também desenvolveram uma inteligência artificial (IA) determinística baseada em uma lista de regras básicas. Os resultados mostram que a inteligência artificial baseada em MCTS obteve um desempenho superior em relação ao agente determinístico.

Diferente de Robilliard, Fonlupt e Teytaud (2014), o trabalho de Pleines (2016), fez uso de uma RNA para criar uma IA capaz de controlar unidades de combate do jogo *StarCraft*. Para isso, foi estruturada uma rede que recebe informações relacionadas ao estado das unidades na entrada e possui a saída projetada para representar a melhor ação a ser tomada. O treino do modelo é realizado de forma supervisionada utilizando uma base de dados contendo o mapeamento entre as melhores ações para cada estado possível. Os resultados mostram que a rede neural desenvolvida é capaz de agir de forma satisfatória em algumas situações, mas não apresentou desempenho suficiente para jogar contra humanos.

Stanescu et al. (2016) desenvolveram Redes Neurais Convolucionais para avaliar estados em jogos *Real Time Strategy* (RTS). Para isso, utilizaram o mesmo objeto de estudo que Pleines (2016), ou seja, o jogo *StarCraft*. A rede neural recebe como entrada detalhes a respeito do estado das unidades durante uma partida e a partir disso fornece como saída a probabilidade de vitória. Os resultados do trabalho mostraram que a técnica utilizada acabou se mostrando superior a outros métodos que possuem o mesmo objetivo.

O trabalho de Lample e Chaplot (2017) teve como objetivo aplicar *Deep Q-Learning* em jogos de tiro em primeira pessoa. Esse foi o primeiro estudo a aplicar o método proposto por (MNIH et al., 2013) em um ambiente 3D com estados parcialmente observáveis. Para isso, foi

utilizada uma Rede Neural Recorrente para jogar *Doom* estimando os valores de Q e escolhendo as melhores ações. Os experimentos mostraram que o método utilizado foi capaz de jogar contra humanos e possui uma boa generalização.

Semelhante a Mnih et al. (2013) e Lample e Chaplot (2017), Xenou, Chalkiadakis e Afantenos (2018) utilizam o algoritmo *Q-Learning* combinado a Redes Neurais Profundas. O objetivo foi aplicar aprendizagem por reforço profundo à ambientes envolvendo jogos de tabuleiro. Eles utilizaram uma Rede Neural Recorrente para obtenção do valor relacionado a qualidade Q de uma determinada ação. O agente desenvolvido conseguiu superar outros agentes com o mínimo de treino.

Świechowski, Tajmajer e Janusz (2018) combinaram MCTS, algoritmo também utilizado por Robilliard, Fonlupt e Teytaud (2014), com algoritmos de aprendizagem supervisionada. A finalidade do trabalho foi melhorar o desempenho de uma IA desenvolvida para o jogo *HearthStone*. Foi utilizada uma Rede Neural que dado um estado do jogo como entrada fornece a probabilidade de vitória de cada jogador. Essas probabilidades são utilizadas para que o MCTS não precise simular um jogo até chegar no término. Os resultados mostram uma IA melhor que outras versões de agentes previamente desenvolvidos e capaz de enfrentar oponentes humanos.

Em toda a literatura, apenas Robilliard, Fonlupt e Teytaud (2014) tiveram o mesmo propósito deste trabalho, que é desenvolver agentes inteligentes para o jogo *7 Wonders*. No entanto, o presente trabalho se difere na metodologia utilizada, pois foram construídos modelos de redes neurais e utilizado *Deep Q-Learning*.

Além disso, pelo propósito de construir agentes inteligentes para um determinado jogo utilizando de redes neurais, este trabalho se assemelha a maioria dos citados nesta seção. Todavia, no que se refere ao objeto de estudo, estes trabalhos tiveram o objetivo de construir agentes para jogos que possuem regras, objetivos e complexidade diferentes do *7 Wonders*.

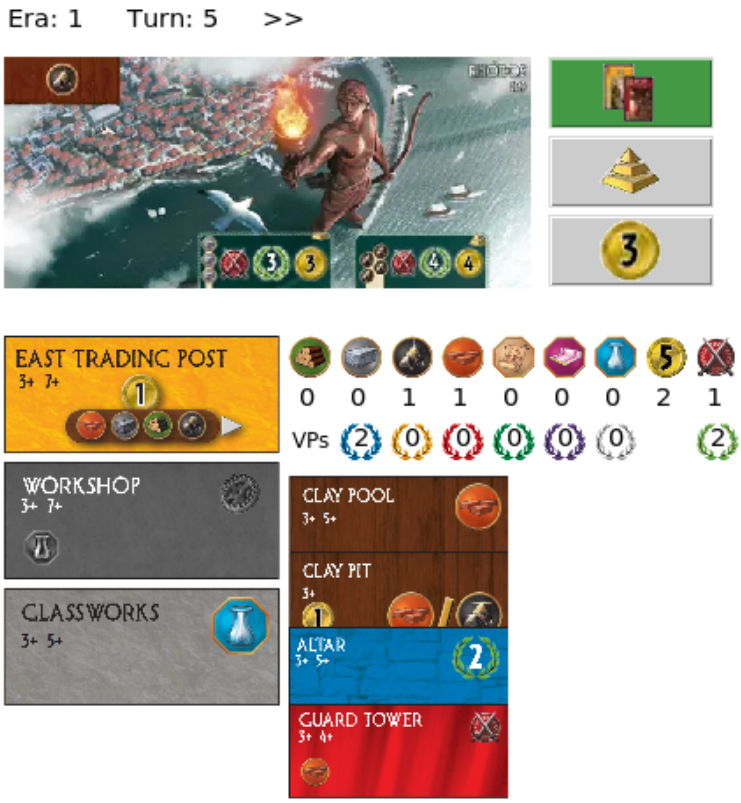
3 DESENVOLVIMENTO

Neste capítulo são descritas as técnicas e ferramentas utilizadas nas etapas de desenvolvimento do presente trabalho. O capítulo está organizado da seguinte forma: inicialmente é feita a descrição da implementação utilizada como ambiente de teste para os agentes implementados. Em seguida é descrito o funcionamento de um agente baseado em regras, um agente baseado em redes neurais e um agente que utiliza a técnica *Deep Q-Learning*.

3.1 AMBIENTE DE TESTE

Em um trabalho prévio (JARDIM et al., 2020), desenvolvemos uma versão digital do jogo *7 Wonders*, mostrada na Figura 3.1, programada utilizando a linguagem *C++*. O intuito disso foi criar um ambiente capaz de executar uma grande quantidade de partidas no menor tempo possível permitindo testar e treinar diferentes implementações de agentes inteligentes de forma fácil e rápida. Os agentes desenvolvidos no presente trabalho foram testados e treinados nesse ambiente.

Figura 3.1 – Interface gráfica do jogo implementado.

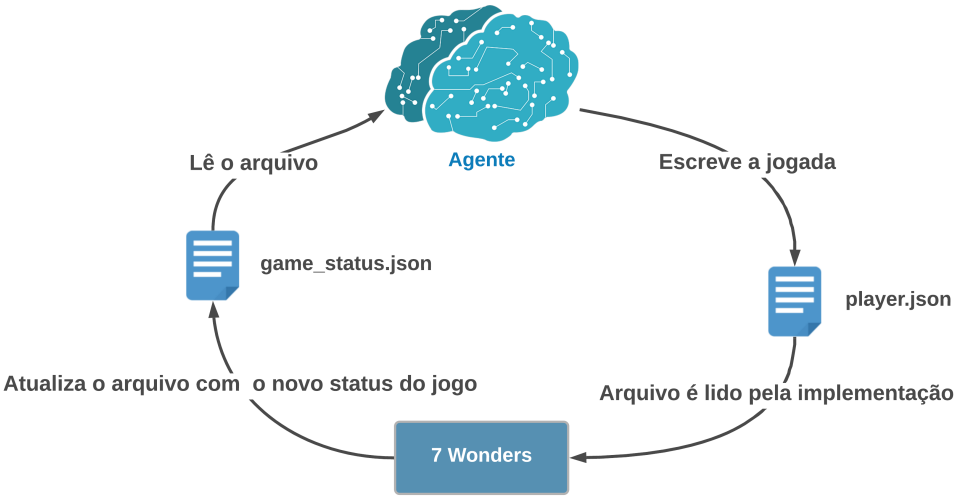


Fonte: Criado pelo próprio autor.

A Figura 3.1 mostra a interface gráfica do jogo que contém o tabuleiro utilizado pelo jogador, botões para escolher uma ação, cartas jogadas, cartas na mão, recursos e pontuação. Os agentes implementados não se comunicam com o jogo por meio dessa interface, mas ela pode ser utilizada para visualizar o que eles jogam.

A troca de informações entre os agentes e o jogo ocorre por meio da leitura e escrita em arquivos no formato *Javascript Object Notation* (JSON). Para isso, são utilizados arquivos nomeados como *player* e *game_status*. Durante a execução de uma partida, o agente faz a leitura do arquivo *game_status* para obter informações relacionadas ao seu estado atual e se baseando nisso escreve a jogada realizada no arquivo *player*. A Figura 3.2 mostra como isso ocorre.

Figura 3.2 – Comunicação entre agente e o jogo 7 Wonders



Fonte: Criado pelo próprio autor.

Durante o início da execução do jogo é criado um arquivo *player* para cada agente. Além disso, o nome do arquivo possui um sufixo, que é o identificador único do agente na partida. Dessa forma, os agentes sabem em qual arquivo devem escrever suas jogadas durante a execução de uma partida.

Além disso, o arquivo *game_status* é atualizado pelo jogo durante a execução de uma partida sempre quando um novo turno inicia e armazena informações relacionadas ao estado atual da partida e dos jogadores, que são:

- Turno atual: números inteiros {0, 1, 2, ..., 20} indicando o turno atual.
- Era atual: números inteiros {1, 2, 3} indicando a era atual.
- Cartas na mão: vetor de números inteiros identificando as cartas na mão.
- Cartas jogáveis: vetor de números inteiros identificando as cartas jogáveis.

- Cartas jogadas: vetor de números inteiros identificando as cartas jogadas.
- Quantidade pontos: número inteiro representando a quantidade total de pontos.
- Quantidade de recursos: cada recurso como minério, pedra, madeira, vidro, tecido, papiro e argila tem suas quantidades representadas por um número inteiro.
- Maravilha utilizada: número inteiro identificando a maravilha.
- Quantidade de estágios construídos na maravilha: número inteiro representando a quantidade de estágios construídos de uma maravilha.

A implementação do *7 Wonders* atua como um servidor que recebe e faz o processamento das jogadas informadas pelos agentes e retorna novas informações relacionadas ao estado de cada jogador na partida. Além disso, o código relacionado à implementação do jogo é independente dos agentes desenvolvidos.

3.2 ESTRUTURA DOS AGENTES

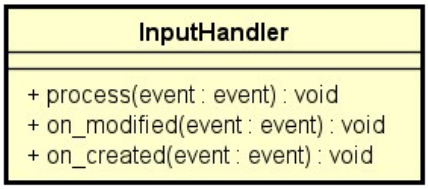
Os agentes desenvolvidos neste trabalho foram programados utilizando a versão 3.7 da linguagem *Python*. A estrutura utilizada para realizar a comunicação com a implementação do jogo *7 Wonders*, brevemente descrita na seção anterior, é semelhante para todos os agentes desenvolvidos. Ela utiliza a biblioteca *watchdog*[1] e a classe *InputHandler*, Figura 3.3, para monitorar mudanças no arquivo *game_status.json*.

Cada agente possui a classe *InputHandler*, que é responsável por monitorar alterações no arquivo *game_status*. Sabendo que o arquivo é alterado pelo jogo somente no início de cada turno, quando uma mudança é detectada, o agente sabe que precisa ler o *game_status* e realizar uma jogada.

As próximas seções descrevem o desenvolvimento de cada um dos 3 agentes. Inicialmente, foi construído um agente baseado em regras para testar a versão digital do *7 Wonders* descrita na Seção 3.1 e criar uma estrutura de comunicação com o jogo para ser utilizada na programação de outros agentes, e, além disso, ser uma primeira IA baseada em trabalhos prévios (ASSUNÇÃO et al., 2019), (JARDIM et al., 2020). Ao decorrer do trabalho foram desenvolvidos um agente que utiliza RNAs treinadas de forma supervisionada e, logo após, um agente que usa um modelo de *Deep Q-Learning*.

[1] https://github.com/gorakhargosh/watchdog

Figura 3.3 – Classe *InputHandler*



Fonte: Criado pelo próprio autor.

3.3 DESENVOLVIMENTO DE UM AGENTE BASEADO EM REGRAS

Em um trabalho prévio, obtivemos o conhecimento de estratégias fortes através da mineração de regras em um grande conjunto de dados relacionados a estatísticas finais de partidas de *7 Wonders* (ASSUNÇÃO et al., 2019). Tais estratégias foram comparadas com um guia[2], chamado *7 Wonders: Ultimate Analysis of the 3-Player game*. Com base nesse guia e no trabalho prévio enumeramos uma lista de pesos que foram atribuídos às cartas.

Obtivemos duas estratégias gerais, enquanto numa os maiores pesos estão na construção de estruturas do tipo militar e civil, na outra é valorizada a obtenção de estruturas científicas. Como sequência do trabalho anterior e como parte deste e do trabalho de Jardim et al. (2020), foi desenvolvido um agente baseado em regras.

O agente desenvolvido utiliza os pesos exemplificados no Quadro 3.1, que são atribuídos de acordo com a condição em que jogador se encontra em uma partida, a maravilha (tabuleiro) e a estratégia abordada. O agente foi programado para jogar de acordo com a estratégia que prioriza a construção de estruturas militares e civis, mas não possui qualquer tipo de aprendizado.

Os pesos são armazenados em uma estrutura de dados chamada de dicionário. Um dicionário é uma coleção não ordenada de pares chave-valor, as chaves dessa coleção são os identificadores únicos das cartas, representados como números inteiros. E os valores são referências para funções que retornam os pesos de cada carta com base no estado do jogador, maravilha e estratégia utilizada. Dessa forma, é possível mapear os pesos para cada carta que o agente possui na mão.

As cartas que o agente possui na mão são representadas na forma de um vetor, que é obtido através da leitura do arquivo *game_status*. Os pesos listados são mapeados para cada carta incluída no vetor. Dessa forma, considerando os recursos disponíveis, a decisão do agente se baseia em escolher a carta jogável que possuir o maior peso. Entende-se por jogável as cartas que podem ser jogadas como estrutura ou construção de um estágio da maravilha. Nos casos em que nenhuma das duas ações está disponível é feito o descarte da estrutura. Além disso, se duas

[2]https://en.boardgamearena.com/forum/viewtopic.php?f=192&t=14557

Quadro 3.1 – Exemplos de regras e pesos.

Carta	Condição	Peso
Campo de Lenhadores	Possuir menos que 2 matérias primas	4
Templo	Ter menos que 2 estruturas civis construídas	3
Templo	Ter pelo menos 2 estruturas civis construídas	1
Quartel	Não ter construído estruturas militares	5
Quartel	Ter construído pelo menos 1 estrutura militar	4
Prensa	Carta é desvalorizada pela estratégia militar	1
Palácio	Carta azul com maior pontuação	5

Fonte: Criado pelo próprio autor.

ou mais ações estiverem disponíveis, a escolha é executada conforme as prioridades mostradas no Quadro 3.2.

Quadro 3.2 – Prioridade das ações em ordem decrescente.

Ação	Prioridade
Construir estágio da maravilha	3
Construir estrutura	2
Descartar	1

Fonte: Criado pelo próprio autor.

Vale ressaltar que o Quadro 3.1 é um mero resumo da lista de pesos que foi construída, pois foram atribuídos pesos para todas as 65 cartas utilizadas em partidas com 3 jogadores. Além disso, cada carta pode possuir mais de um peso, que é atribuído de acordo com o contexto de uma partida e a maravilha na qual o agente possui, exemplo disso são os pesos atribuídos às cartas *quartel* e *templo* mostradas no Quadro 3.1.

Este agente baseado em regras é capaz de empregar as principais estratégias obtidas a partir da análise das estatísticas dos melhores jogadores no *ranking* do BGA (*Board Game Arena*)[3]. As estratégias se aplicam a grande parte das situações em partidas com 3 jogadores e podem até mesmo fazer com que o agente tenha um desempenho considerável. Porém, o agente não aprende ao longo das partidas, o que limita o seu desempenho e torna suas jogadas mais previsíveis.

3.4 DESENVOLVIMENTO DE UM AGENTE QUE UTILIZA REDES NEURAIS

Para jogar contra o agente baseado em regras descrito na Seção 3.3, também foi desenvolvido neste trabalho um agente para jogar partidas de *7 Wonders* com 3 jogadores utilizando RNAs, que foram treinadas de forma supervisionada. Isso foi feito com base em um conjunto

[3]https://boardgamearena.com/

de dados que contém o histórico de 1000 partidas jogadas por 3 instâncias do agente baseado em regras. Foram considerados apenas os históricos das instâncias relacionadas aos agentes vencedores em cada partida, com o intuito de obter as melhores ações.

A comunicação entre o agente desenvolvido e o ambiente do jogo foi feita seguindo a mesma estrutura descrita na Seção 3.2. Além disso, o agente utiliza 3 redes neurais, sendo uma para cada *era* específica da partida. Isso foi programado utilizando uma estrutura condicional que durante a execução do agente determina qual rede neural será instanciada com base na *era* em que a partida se encontra.

As redes desenvolvidas para serem utilizadas com o agente descrito nessa seção são do tipo MLP e foram desenvolvidas com o *framework PyTorch*[4] na plataforma *Google Colab*[5]. Como não havia nenhuma outra IA baseada em redes neurais para o *7 Wonders*, era esperado que o primeiro passo fosse explorar a utilização de redes neurais comuns e criar uma arquitetura inicial que possa ser expandida futuramente.

3.4.1 Conjunto de dados

O conjunto de dados que contém o histórico das partidas de *7 Wonders* jogadas pelos agentes armazena dados relacionados a estados e jogadas realizadas ao decorrer de 1000 partidas. Esses dados são armazenados em arquivos CSV e possuem as colunas mostradas no Quadro 3.3.

As colunas *carta* 1 a 7 representam todas as cartas que o agente possuía na mão, as demais são relacionadas a recursos, estruturas construídas e informações gerais de uma partida, como *era* e *ID* da maravilha utilizada. Considerando a fase de treino de RNAs, essas colunas foram utilizadas como entradas para as redes desenvolvidas com exceção das colunas *carta jogada* e *ação*, que são as saídas desejadas para o modelo.

3.4.2 Estruturas das Redes Neurais Artificiais construídas

Uma partida de *7 Wonders* possui 3 *eras* e para cada uma delas variam a quantidade e disponibilidade de cartas e jogadas. Para a primeira *era* existem 21 cartas e 63 jogadas possíveis, a segunda *era* possui a mesma quantidade de jogadas e cartas que a primeira, já na terceira são 26 cartas e 78 jogadas possíveis. Neste primeiro modelo, foi optado por usar 3 RNAs sendo que cada uma é responsável por jogar uma determinada *era* de uma partida.

Durante uma partida, as RNAs projetadas recebem como entrada o estado atual do jogador, obtido pela leitura do arquivo *game_status* e produzem na saída uma jogada. O estado atual

[4]https://pytorch.org/

[5]https://colab.research.google.com/

Quadro 3.3 – Colunas do conjunto de dados relacionado ao histórico de partidas.

Nome da Coluna	Tipo
Carta 1	Numérico (Identificador único)
...	Numérico (Identificador único)
Carta 7	Numérico (Identificador único)
Quantidade de estruturas civis	Numérico
Quantidade de estruturas militares	Numérico
Quantidade de estruturas comerciais	Numérico
Quantidade de estruturas científicas	Numérico
Quantidade de cartas de matéria prima	Numérico
Quantidade de cartas de produtos manufaturados	Numérico
Quantidade de cartas de guilda	Numérico
ID da Maravilha	Numérico (Identificar Único)
Estágio da Maravilha	Numérico
Quantidade de Escudos	Numérico
Quantidade de moedas	Numérico
Era	Numérico
Carta jogada	Numérico (Identificador Único)
Ação	Textual

Fonte: Criado pelo próprio autor.

do jogador é composto pelas mesmas colunas mostradas no Quadro 3.3, com exceção de *carta jogada* e *ação*. Além disso, para representar as cartas na mão do jogador foram necessários neurônios de entrada para representar todas as cartas possíveis, dependendo da *era* para a qual a rede neural foi projetada.

A representação das cartas na mão do agente foi feita da seguinte maneira: a rede desenvolvida para jogar a primeira *era* de uma partida possui 21 neurônios de entrada que representam todas as cartas possíveis, já a rede da segunda e terceira *era* possuem 21 e 26 neurônios respectivamente. Os neurônios que correspondem às cartas que estão na mão do agente recebem o valor 1 e ao restante é atribuído o valor 0.

Os neurônios da camada de saída de cada RNA representam as jogadas possíveis em uma determinada *era*, ou seja, as redes neurais projetadas para jogar a primeira, segunda e terceira *era* de uma partida possuem 63, 63 e 78 neurônios respectivamente. Para que essa representação funcionasse, foi necessário codificar o par carta e ação, que constituem uma jogada, para serem representados nas saídas da RNA como valores únicos.

Durante uma partida de *7 Wonders*, o agente deve escolher uma carta, que é representada por um número de identificação (*ID*) e uma ação relacionada a carta, que também pode ser representada como um valor numérico. Os Quadros 3.4 e 3.5, mostram identificadores para as cartas e ações respectivamente.

Para que a informação da carta selecionada e a ação realizada fossem representadas utilizando um único valor, foi utilizado o cálculo $A \times 100 + C$ onde A é o identificador numérico

Quadro 3.4 – Exemplos de cartas de identificadores

Identificador (ID)	Carta
3	Argileira
16	Vidraçaria
54	Apotecário

Fonte: Criado pelo próprio autor.

Quadro 3.5 – Ações e seus identificadores

Identificador (ID)	Ação
1	Construir carta
2	Construir estágio da maravilha
3	Descartar

Fonte: Criado pelo próprio autor.

da ação e C o identificador da carta. Por exemplo, a jogada Construir Argileira seria codificada como 103.

Para recuperar os valores do par ação e carta, utilizado para informar a jogada através da escrita no arquivo *player_status*, basta obter a divisão inteira por 100, para recuperar a ação, e o resto inteiro da divisão por 100 para obter a carta.

Dessa forma, o agente desenvolvido utiliza 3 redes neurais do tipo MLP, uma para cada *era* do jogo, com entradas compostas pelas cartas na mão e estado atual do jogador e neurônios de saída que representam as jogadas possíveis em uma *era* específica de uma partida. As quantidades de neurônios de entrada e saída para cada rede são mostradas no Quadro 3.6.

Quadro 3.6 – Estrutura das entradas e saídas das redes neurais desenvolvidas.

Era	Quantidade de neurônios de entrada	Quantidade de neurônios de saída
1	32	63
2	32	63
3	37	78

Fonte: Criado pelo próprio autor.

3.4.3 Treino das redes neurais

Foram treinadas diferentes configurações de RNAs com o intuito de encontrar as redes com melhores acurácias de treino e teste. Isso foi feito considerando a estrutura mencionada na Seção 3.4.2, que utiliza uma RNA para cada *era* de uma partida do *7 Wonders*. Para isso, os dados de treino e teste foram separados por *era*. Alguns dos hiperparâmetros das redes treinadas durantes os testes são mostrados no Quadro 3.7.

Quadro 3.7 – Alguns hiperparâmetros testados para as redes.

Hiperparâmetro	Rede era 1	Rede era 2	Rede era 3
Entradas	32	32	37
Saídas	63	63	78
Otimizador	*Adam*	*Adam*	*Adam*
Camadas Intermediárias (C.I.)	2 a 4	2 a 4	2 a 4
Quantia de neurônios nas C.I.	70 a 95	70 a 95	70 a 95
Função de Perda	*Cross Entropy*	*Cross Entropy*	*Cross Entropy*
Taxa de Aprendizado	0,0001 e 0,00001	0,0001 e 0,00001	0,0001 e 0,00001

Fonte: Criado pelo próprio autor.

Além dos hiperâmetros mostrados no Quadro 3.7, foram testadas diferentes funções de ativação para os modelos de RNAs construídos. Nas camadas intermediárias foram testadas as funções Sigmoide, ReLu (*Rectfied Linear Uniform*), *Leaky ReLu* e Tanh (Tangente Hiperbólica). Na camada de saída é utilizada a função *Softmax*, que é utilizada em problemas de classificação que envolvem várias classes, mas para esse problema tem o propósito de fornecer a probabilidade de uma ação ser a correta, pois nesse caso podemos imaginar as classes como sendo as ações.

As redes que obtiveram melhor resultado no treinamento são mostradas no Quadro 3.8 e suas acurácias são apresentadas no Quadro 3.9. É notável a ocorrência de um *overfitting*, ou seja, a RNA ficou super-ajustada ao conjunto de dados relacionado ao treino. Isso ocorre porque os dados usados para alimentar a rede neural, durante o treino, são o histórico de partidas jogadas pelo agente baseado em regras descrito na Seção 3.3 e devido as estratégias seguidas pelo agente, o conjunto de dados fica desbalanceado, pois são priorizadas determinadas ações tornando-as mais frequentes que outras.

Quadro 3.8 – Redes que obtiveram os melhores resultados.

Rede neural	Função de ativação	Taxa de aprendizado	Camadas intermediárias (c x n)[6]
Rede era 1	ReLu	0,0001	2 x 80
Rede era 2	ReLu	0,0001	2 x 80
Rede era 3	ReLu	0,0001	2 x 90

Fonte: Criado pelo próprio autor.

Nesse contexto, o *overfitting* que ocorre acaba não sendo um problema, pois o objetivo é fazer que a RNA aprenda com as vitórias do agente baseado em regras e utilize uma estratégia semelhante. Sendo assim, as 3 melhores redes, tendo como critério a acurácia de treino, foram utilizadas no agente desenvolvido. Os modelos treinados foram salvos para que fossem carregados pelo agente durante a execução das partidas.

[6] A letra c representa a quantidade de camadas intermediárias e n está relacionado ao número de neurônios em cada camada.

Quadro 3.9 – Acurácias obtidas no treinamento.

Rede neural	Épocas de Treino	Acurácia de treino	Acurácia de teste
Rede era 1	5000	0,9714	0,4623
Rede era 2	5000	0,9988	0,4000
Rede era 3	5000	0,9998	0,5257

Fonte: Criado pelo próprio autor.

3.4.4 Execução do agente

Ao iniciar a execução, é feito o carregamento dos modelos treinados para 3 redes neurais que são instanciadas no agente, sendo uma rede para cada *era* da partida. Durante uma partida, são extraídas as informações do arquivo *game_status*, relacionadas as entradas da rede, que são passadas para a mesma. A saída da rede é decodificada para que sejam obtidas a carta e ação que serão informadas no arquivo *player*.

Para evitar que eventuais jogadas inválidas fossem realizadas pelo agente, durante uma partida, as escolhas consideram apenas os neurônios que representam as jogadas válidas com maior probabilidade. Isso garante que o agente implementado escolha apenas as jogadas que são válidas.

O agente implementado é capaz de empregar, as mesmas estratégias obtidas pelo agente baseado em regras descrito na Seção 3.3, por isso, se espera que obtenha um desempenho muito semelhante ao desse agente. Vale lembrar que o agente descrito nessa seção, foi treinado com dados relacionados a partidas em que o agente baseado em regras venceu seguindo estratégias que se aplicam a maioria das situações no *7 Wonders* e são recomendadas por um especialista[7].

Além disso, as redes implementadas nesta seção constituem o primeiro passo utilizando redes neurais. A próxima Seção 3.5, descreve o desenvolvimento de um terceiro agente inteligente, onde foi utilizada uma técnica sofisticada de aprendizagem por reforço profundo, que torna o agente capaz de aprender enquanto joga as partidas.

3.5 DESENVOLVIMENTO DE UM AGENTE COM *DEEP Q-LEARNING*

Para o desenvolvimento do terceiro e último agente implementado neste trabalho, foi utilizada uma técnica de aprendizagem por reforço profundo chamada de *Deep Q-Learning*. Durante a execução de uma partida, os estados do jogador são passados a uma RNA, que foi implementada com o *PyTorch*[8], para estimar os valores de Q relacionados a todas as ações possíveis. As jogadas do agente se baseiam em escolher as ações que possuem o maior valor.

[7] https://en.boardgamearena.com/forum/viewtopic.php?f=192&t=14557
[8] https://pytorch.org/

3.5.1 Estrutura da rede neural utilizada

A RNA utilizada neste agente possui 85 neurônios de entrada e 195 neurônios na saída. Além disso, possui 4 camadas intermediárias, cada uma com 128 neurônios. Esta é uma rede semelhante às redes descritas na Seção 3.4, pois ao decorrer da partida são fornecidas entradas que são os estados do agente para obter na saída uma ação. No entanto, neste caso a rede neural atua como um estimador dos valores Q, ou *qualidade* das ações, ao invés de fornecer a probabilidade uma determinada ação ser a correta em um determinado estado.

A principal diferença entre o agente descrito nesta seção e o agente descrito na Seção 3.4 está na abordagem utilizada. Enquanto o agente desenvolvido na Seção 3.4 é treinado de forma supervisionada utilizando um conjunto de dados, o agente desenvolvido com a técnica de *Deep Q-Learning* é treinado para maximizar as recompensas que recebe enquanto explora o jogo, jogando diversas partidas de treino.

Vale mencionar que este agente utiliza apenas uma rede neural para jogar uma partida inteira, ao contrário do agente anterior, que possui 3 redes neurais, sendo uma para cada *era* de uma partida. Além disso, alguns atributos de entrada utilizados para representar a quantidade de estruturas que o agente possui na partida foram substituídos por atributos que representam a quantidade de cada tipo de recurso que o agente dispõe, já que essas estruturas são diretamente ligadas a esses recursos, o Quadro 3.10 mostra os novos atributos de entrada.

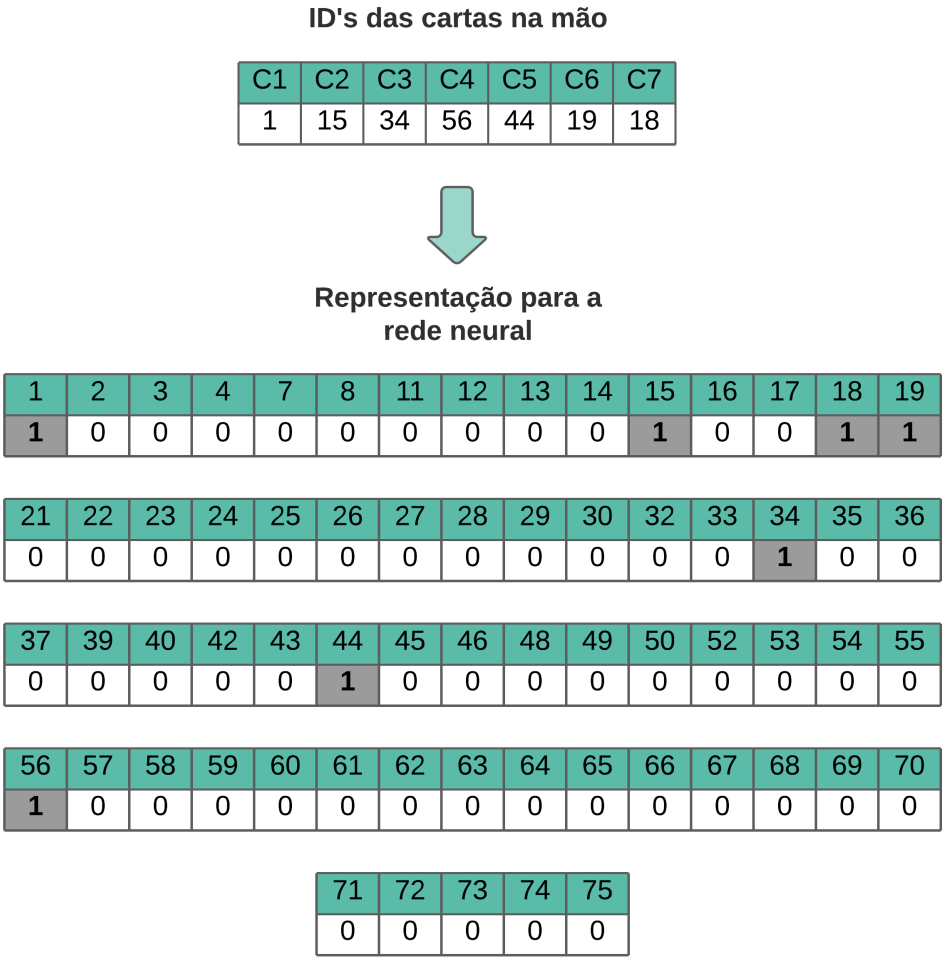
Quadro 3.10 – Atributos de entrada da rede neural.

Atributo	Tipo
Cartas na mão	Vetor de número inteiros
Quantidade de estruturas civis	Número inteiro
Quantidade de estruturas militares	Número inteiro
Quantidade de estruturas comerciais	Número inteiro
Quantidade de cartas de guilda	Número inteiro
Quantidade de madeira	Número inteiro
Quantidade de minério	Número inteiro
Quantidade de argila	Número inteiro
Quantidade de pedra	Número inteiro
Quantidade de vidro	Número inteiro
Quantidade de tecido	Número inteiro
Quantidade de papiro	Número inteiro
Quantidade de engrenagens	Número inteiro
Quantidade de *tablets*	Número inteiro
Quantidade de compassos	Número inteiro
Quantidade de escudos	Número inteiro
Quantidade de moedas	Número inteiro
ID da maravilha	Número inteiro
Estágios construídos da maravilha	Número inteiro
Era	Número inteiro

Fonte: Criado pelo próprio autor.

Para representar as *cartas na mão* do jogador foi aplicada a mesma técnica de representação utilizada pelas redes neurais descritas na Seção 3.4, são utilizados 65 neurônios de entrada para representar todas as cartas que possivelmente podem aparecer na mão do agente durante uma partida, dessa forma as cartas que fazem parte da mão do jogador recebem valor 1 e o restante 0, isso é mostrado na Figura 3.4.

Figura 3.4 – Representação das cartas na mão do agente para a rede neural.



Fonte: Criado pelo próprio autor.

Quanto às saídas da RNA desenvolvida, elas correspondem ao valor Q de todas as ações que podem ser realizadas durante uma partida de 3 jogadores. Em uma partida de *7 Wonders* podem ser selecionadas 65 cartas diferentes, que são obtidas aleatoriamente do baralho. Considerando que para cada carta existem 3 ações que são, construir carta, descartar e construir estágio da maravilha, se obteve 195 (65×3) ações possíveis.

Além disso, para representação das ações na saída da RNA foi utilizada a mesma codificação descrita na Seção 3.4, que codifica uma ação (jogada) como $A \times 100 + C$, onde A é uma ação realizada sobre uma carta e C, o identificador da carta. A recuperação do valor da carta e da ação é feita através da divisão e do resto inteiro da divisão, que fornecem o identificador da ação e da carta, respectivamente.

Após a definição da estrutura da rede neural, o passo seguinte foi a definição das recompensas que seriam fornecidas ao agente, para que ele aprendesse quais ações poderiam fornecer vantagens ou desvantagens durante uma partida.

3.5.2 Recompensas

A pontuação total no *7 Wonders* é calculada pela soma dos pontos de vitória fornecidos por estruturas que são representadas por cartas, vitórias em conflitos militares e moedas. Se a recompensa fornecida fosse baseada apenas na pontuação, poderiam haver muitas etapas até que fosse recebida alguma recompensa, pois muitas cartas não fornecem pontos diretamente, isso dificultaria a avaliação do agente para aprender boas ações e aumentaria consideravelmente o tempo de treino, pois as recompensas na memória de *replay* do agente se tornariam muito esparsas. Lample e Chaplot (2017) descrevem esse problema e utilizam uma técnica proposta por Ng e Jordan (2003) chamada *reward shaping*.

A técnica consiste em fornecer, além da pontuação total, recompensas intermediárias. Considerando isso, as recompensas fornecidas ao agente se basearam na lista de pesos utilizada para o agente baseado em regras, descrito na Seção 3.4. Essa lista é novamente exemplificada no Quadro 3.11. Esses pesos são somados a pontuação recebida por uma jogada e o valor obtido é utilizado como a recompensa do agente.

Quadro 3.11 – Lista de pesos.

Carta	Condição	Peso
Estátua	Se tiver construído menos que 2 estruturas	3
Farol	Se tiver construído pelo menos 4 estruturas comerciais	3
Farol	Se tiver construído menos que 4 estruturas comerciais	1
Torre de guarda	Não ter construído estruturas militares	5
Torre de guarda	Ter construído pelo menos 1 estrutura militar	4
Câmara de comércio	Possuir mais que 2 bens manufaturados	3
Panteão	Carta azul que fornece 7 pontos de vitória	5

Fonte: Criado pelo próprio autor.

Além disso, considerando que o descarte de muitas cartas diminui consideravelmente a pontuação total em uma partida, é fornecida uma pontuação negativa para o agente quando ele descarta mais que 2 cartas em uma *era*. Isso foi feito para que o agente aprendesse a mais rapidamente, gerenciar os recursos e para que explorasse mais ações relacionadas a construção de estruturas.

O parâmetro gama γ, que é o fator de desconto aplicado a recompensas futuras, foi definido como 0,99. Com a ideia de que o agente valorizasse mais as recompensas fornecidas pela construção de recursos e estruturas que levam a obtenção de maiores pontuações a longo prazo. A forma como o agente explora o ambiente para obter essas recompensas é descrita a

seguir.

3.5.3 Exploração do ambiente

O agente desenvolvido explora o jogo por meio de ações aleatórias ou considerando os valores de Q fornecidos pela rede neural que são associados a cada ação. Para isso, é utilizado um parâmetro épsilon ε, que é comparado a um valor gerado aleatoriamente, se o valor gerado for maior que ε, o agente realiza a ação com maior valor de Q atual, caso contrário, a ação realizada é aleatória. Isso descreve o algoritmo *Epsilon-Greedy*, utilizado em aprendizagem por reforço para balancear exploração e exploitação.

O valor de épsilon ε foi inicializado com valor 1 e recebe um decremento de 0,003 a cada turno, até atingir o valor mínimo de 0,01. Dessa forma, o agente começa o treino realizando muitas ações aleatórias, com o intuito de explorar diferentes jogadas e as recompensas recebidas por realizá-las. Conforme o treino avança, o valor de ε diminui e o agente passa a utilizar uma política gulosa para a seleção de ações, ou seja, realiza as ações com maior valor Q estimado pela rede, mas nunca deixando de explorar novas jogadas.

A experiência obtida pelo agente durante as partidas do *7 Wonders* é armazenada em uma estrutura chamada de *replay memory*. Essa estrutura armazena os estados, ações, recompensas recebidas, estados seguintes e estados finais. Os detalhes de como isso foi implementado são descritos a seguir.

3.5.4 *Replay* de Experiência

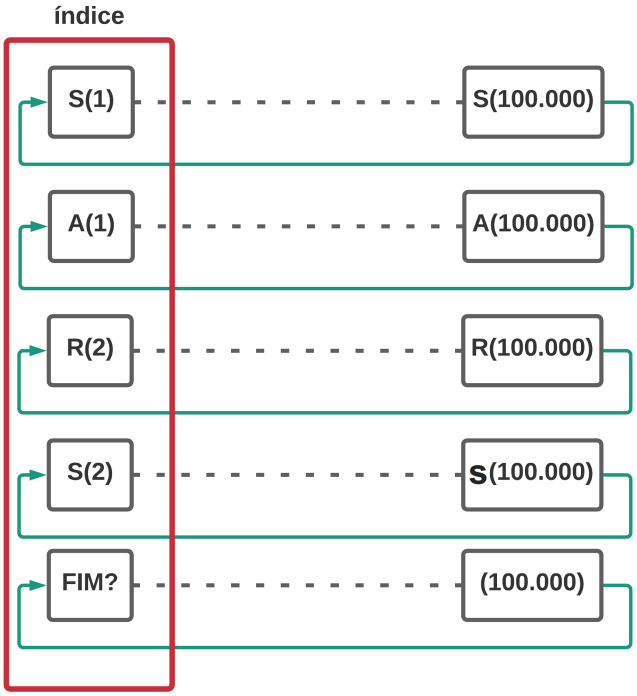
A memória de *replay* é representada por meio de 5 vetores, cada vetor armazena o valor de um atributo da tupla formada pelo estado do agente S_t, jogada A_t, recompensa R_{t+1}, próximo estado S_{t+1} e por um valor booleano que indica se o estado observado é o último de uma partida. Os vetores possuem tamanho 100.000, e são acessados por meio de um índice circular. A estrutura da memória de replay é exemplificada na Figura 3.5.

Depois de definir a estrutura da rede neural, as recompensas e como as experiências do agente seriam armazenadas para que a rede neural pudesse ser treinada. A próxima etapa deste trabalho envolveu, obviamente, o treino do agente.

3.5.5 Treino do agente

Pelo fato deste ser o primeiro trabalho que aplica *Deep Q-Learning* ao *7 Wonders*, foram definidos hiperparâmetros com valores comumente utilizados em outros exemplos de aplica-

Figura 3.5 – Estrutura da memória de *replay* do agente desenvolvido.



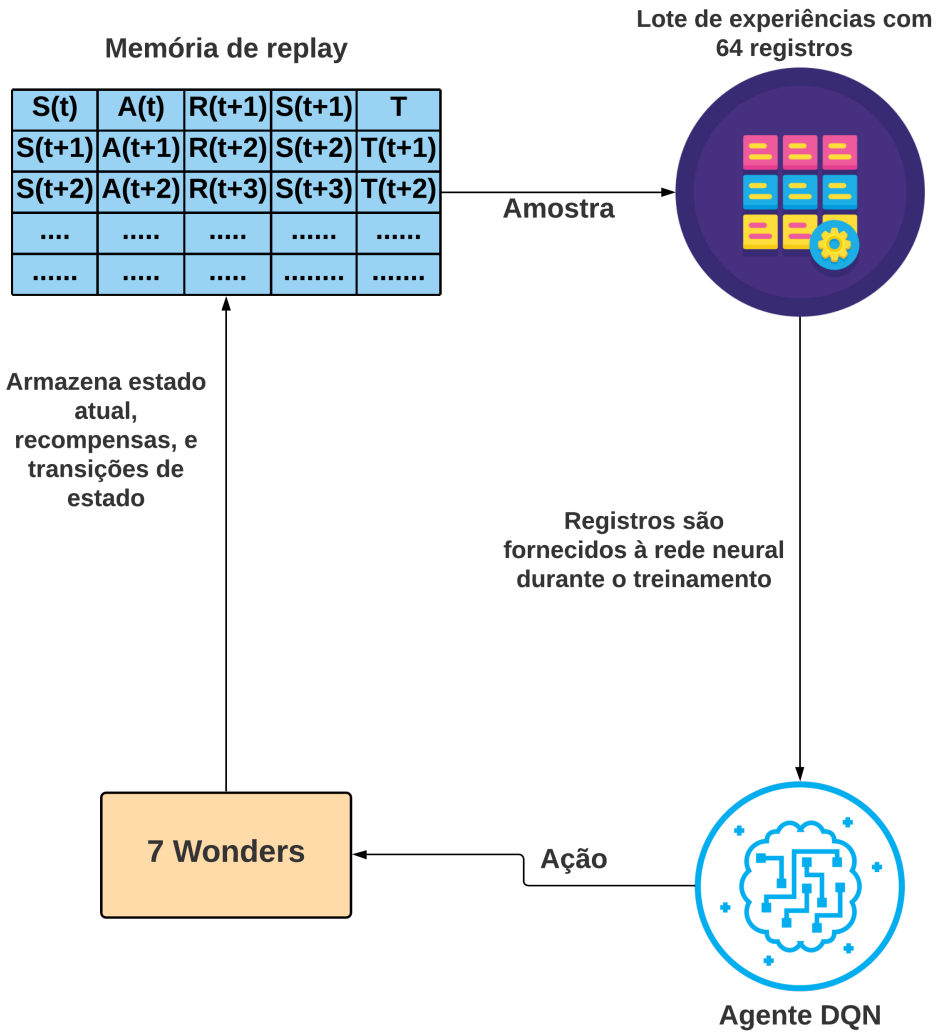
Fonte: Criado pelo próprio autor.

ções. Nas camadas escondidas da RNA, foi utilizada a função *ReLu*, a taxa de aprendizado foi definida com o valor 0,0001, e a função de custo e otimizador escolhidos, foram MSE e *Adam*, respectivamente. Depois da definição desses hiperparâmetros, se deu início ao treino do agente.

Durante o treino do agente, que ocorreu a cada etapa, ou seja, a cada turno das partidas que foram jogadas, são extraídos lotes com 64 registros selecionados aleatoriamente da memória de *replay*. Esses registros contém as ações, transições de estados e recompensas e são utilizados para que a rede neural seja treinada para aproximar os valores de Q para cada ação dado o estado do agente na partida. Esse processo é resumido na Figura 3.6.

Para treinamento, o agente jogou 5000 partidas de *7 Wonders*, para 3 jogadores, contra 2 instâncias de um agente programado para fazer apenas jogadas aleatórias. Os resultados obtidos para o agente baseado em *Deep Q-Learning*, descrito nessa seção, e para os demais agentes desenvolvidos no presente trabalho são analisados e discutidos no próximo capítulo.

Figura 3.6 – Processo de treino do agente DQN resumido.



Fonte: Criado pelo próprio autor.

4 RESULTADOS E CONSIDERAÇÕES

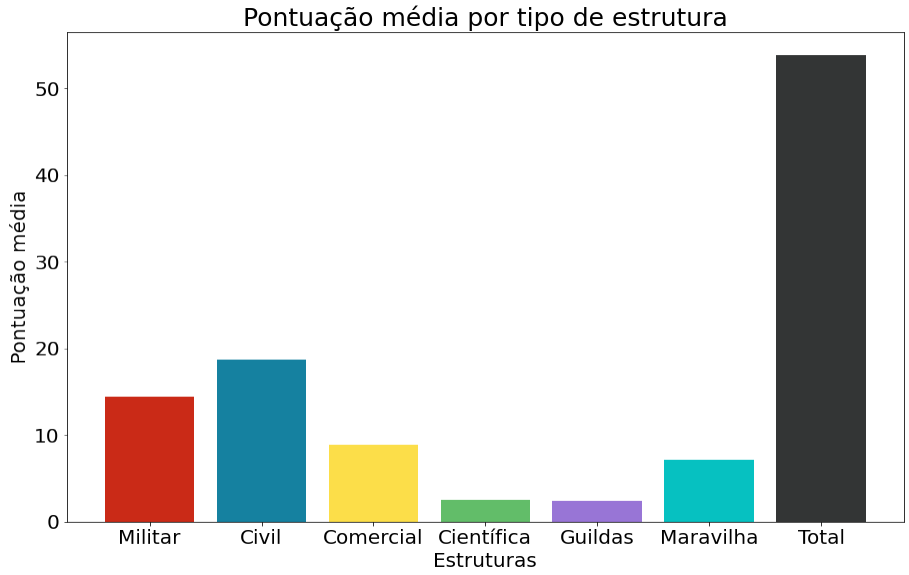
Todos os agentes foram desenvolvidos para jogar a versão de 3 jogadores do *7 Wonders* e foram inicialmente testados em jogos contra um agente simples programado para fazer apenas jogadas aleatórias. Não foi possível testar os agentes contra humanos, pois o *Board Game Arena* (BGA)[1], a única plataforma *online* que fornece uma versão *multiplayer* e popular do jogo *7 Wonders*, não disponibiliza uma *API* para permitir que agentes inteligentes joguem contra humanos.

Todos os testes foram realizados em um computador com 16 *GB* de memória *RAM*, processador *Ryzen 5 1400 3.4 Ghz* e sistema operacional *Ubuntu*. Os primeiros testes foram realizados com o agente baseado em regras descrito no capítulo anterior, para que fossem comparados com os resultados dos demais agentes desenvolvidos neste trabalho.

4.1 TESTES COM O AGENTE BASEADO EM REGRAS

O agente baseado em regras jogou 1000 partidas contra 2 instâncias de um agente programado para fazer apenas jogadas aleatórias. Foi observada a pontuação média obtida para cada tipo de estrutura do jogo e a pontuação total média. O Gráfico 4.1 mostra as pontuações médias obtidas pelo agente para cada tipo de estrutura nas 1000 partidas jogadas.

Gráfico 4.1 – Pontuação média obtida pelo agente baseado em regras para cada tipo de estrutura.



Fonte: Próprio autor.

Com base no Gráfico 4.1 pode-se observar que o agente baseado em regras obtém a

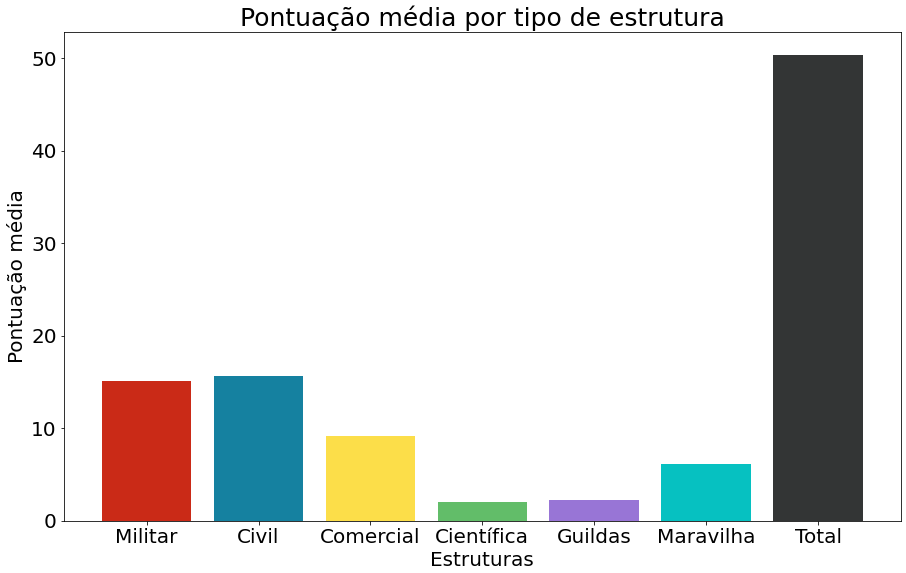
[1] https://boardgamearena.com

maior parte dos pontos através da construção de estruturas civis e militares, o que era esperado, pois suas regras se baseiam numa estratégia que prioriza a construção de cartas civis e militares durante uma partida. Além disso, o tempo médio para a realização de uma jogada foi de 4,55 milissegundos e a pontuação total média foi de 53,83 pontos, é interessante mencionar que a pontuação média dos melhores jogadores do BGA é de 53 pontos.

4.2 TESTES COM O AGENTE BASEADO EM REDES NEURAIS

Os testes realizados para o agente que utiliza redes neurais possui os mesmos critérios utilizados para o teste do agente baseado em regras. O agente que utiliza redes neurais jogou 1000 partidas contra 2 instâncias de um agente que faz jogadas aleatórias. O Gráfico 4.2 além da pontuação total média, mostra a pontuação média obtida para cada tipo de estrutura.

Gráfico 4.2 – Pontuação média obtida pelo agente baseado em redes neurais para cada tipo de estrutura.



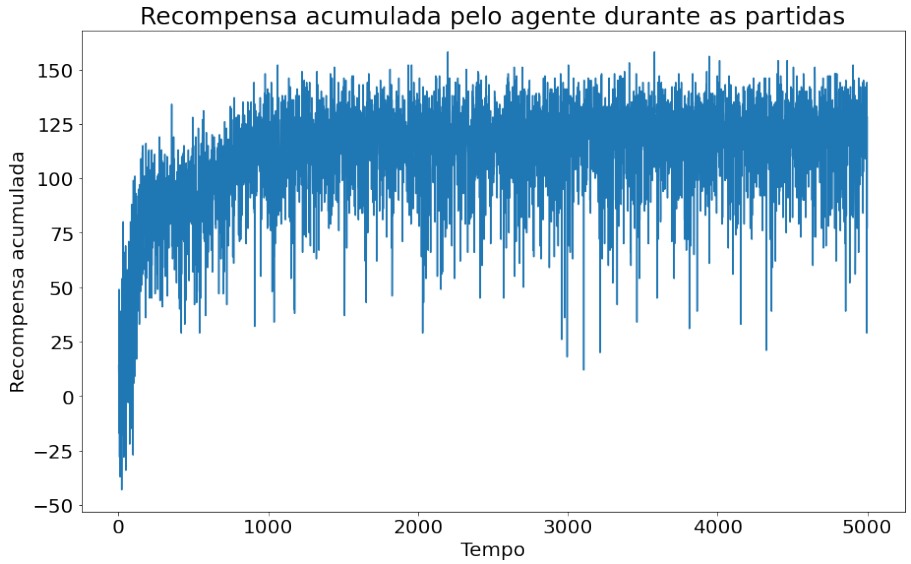
Fonte: Próprio autor.

Os resultados do agente baseado em redes neurais, obviamente, foram muito semelhantes aos do agente baseado regras, pois as redes neurais foram treinadas com dados das melhores partidas do agente baseado em regras. A partir do Gráfico 4.2 pode ser observado que a maior parcela dos pontos é obtida através da construção de estruturas militares e civis, isso significa que as redes neurais aprenderam a jogar utilizando a mesma estratégia que o agente baseado em regras. Além disso, a pontuação total média obtida pelo agente foi de 50,35 pontos e o tempo médio para fazer uma jogada foi de 3,87 milissegundos.

4.3 TESTES DO AGENTE DESENVOLVIDO COM *DEEP Q-LEARNING*

O agente que foi desenvolvido utilizando a técnica de aprendizagem por reforço profundo chamada de *Deep Q-Learning* foi primeiramente treinado jogando 5000 partidas contra 2 instâncias do agente que realiza jogadas aleatórias. O Gráfico 4.3 tem por objetivo mostrar as recompensas acumuladas pelo agente ao longo das 5000 partidas jogadas.

Gráfico 4.3 – Recompensa acumulada pelo agente desenvolvido com *Deep Q-Learning* ao longo de 5000 partidas.



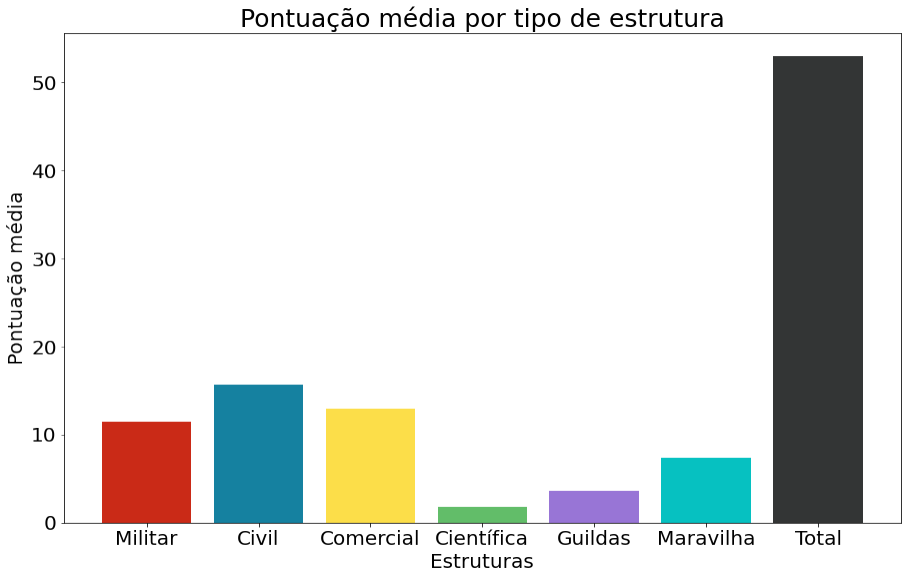
Fonte: Próprio autor.

Ao observar o Gráfico 4.3 é notável a grande variação no acúmulo das recompensas obtidas pelo agente ao longo do treino. Isso se deve ao fato de que os pesos que foram utilizados no desenvolvimento do agente baseado regras foram utilizados como parte das recompensas fornecidas ao agente baseado em *Deep Q-Learning*. Dessa forma, quando o agente não consegue obter as cartas que são priorizadas pela estratégia, a recompensa acumulada acaba diminuindo. Além disso, as cartas do baralho são obtidas de forma aleatória ao longo das *eras* de uma partida, portanto, é natural que hajam partidas em que o agente não consegue as cartas que fornecem as melhores recompensas.

Depois de ser treinado ao longo de 5000 partidas, para fins de comparação com os demais agentes desenvolvidos neste trabalho, o agente baseado em *Deep Q-Learning* jogou mais 1000 partidas. Foi calculada a pontuação média total e a pontuação média obtida pela construção de cada tipo de estrutura do jogo, isso é mostrado no Gráfico 4.4. Além disso, vale lembrar que os cálculos foram realizados considerando apenas as 1000 partidas após o treino.

Com base no Gráfico 4.4 pode-se dizer que o agente baseado em *Deep Q-Learning* obtém a maior parte dos pontos construindo estrutura civis, militares e comerciais, o que era esperado. Isso se deve ao fato de que parte das recompensas são baseadas em uma estratégia que tem como prioridade a construção de estruturas civis e militares. Além disso, diferente

Gráfico 4.4 – Pontuação média obtida pelo agente baseado em *Deep Q-Learning* para cada tipo de estrutura.



Fonte: Próprio autor.

dos demais, o agente priorizou a construção de estruturas comerciais, isso ocorreu porque é dada uma punição negativa para o descarte excessivo durante uma *era* da partida, dessa forma, o agente aprendeu a ter recursos por meio de moedas de comércio para conseguir construir as estruturas e evitar o descarte.

Além disso, pontuação média total obtida pelo agente baseado em *Deep Q-Learning* foi de 52,97 pontos. E o tempo médio para realizar uma jogada foi de 7,21 milissegundos. A próxima seção descreve testes realizados em partidas que os agentes desenvolvidos neste trabalho jogaram uns contra os outros.

4.4 TORNEIO ENTRE OS AGENTES DESENVOLVIDOS

Todos os agentes desenvolvidos neste trabalho jogaram 500 partidas entre si, os resultados são mostrados na tabela 4.1. Pode-se dizer que as pontuações médias dos agentes ficaram equilibradas. Com relação a contagem de vitórias, os agentes que obtiveram os melhores resultados foram o baseado em regras e o baseado em redes neurais.

O agente que foi desenvolvido com a técnica de *Deep Q-Learning*, ficou atrás dos outros agentes nos resultados, porque como visto anteriormente, comparando com a pontuação média obtida pela construção de estruturas militares, ele preza mais pelas estruturas comerciais. Dessa forma, perde pontos militares na etapa de conflitos e acaba beneficiando os adversários, que ganham esses pontos. Por mais que ao jogar contra os outros agentes, ele não tenha obtido um resultado melhor, não significa que a técnica utilizada seja pior que as demais. Devido as

Tabela 4.1 – Tabela com o resultado de 500 partidas jogadas entre os agentes desenvolvidos.

Agente	Pontuação média	Contagem de vitórias	Porcentagem de vitórias
Baseado em regras	42,92	193	38,6
Redes neurais	42,76	199	39,8
Deep Q-Learning	39,09	108	21,6

Fonte: Próprio autor.

restrições de tempo, e tendo em vista que a técnica demanda um tempo considerável para treino e ajuste de hiperparâmetros, não foi possível fazer muitos testes e dedicar mais tempo para o treinamento do agente.

5 CONCLUSÃO

O objetivo deste trabalho foi desenvolver agentes inteligentes para o jogo de tabuleiro *7 Wonders*. Para isso, foram desenvolvidos 3 agentes e cada agente utiliza uma técnica diferente. O primeiro é um agente simples programado para fazer jogadas baseadas numa lista de pesos e para servir de base para o desenvolvimento de outros agentes. Os demais agentes desenvolvidos utilizam técnicas relacionadas a redes neurais.

O *7 Wonders* está entre os jogos de tabuleiro mais premiados e populares do mundo. Além disso, é um jogo complexo que não possui trabalhos de inteligência artificial conhecidos. Este trabalho faz parte de uma sequência de trabalhos, que visam o desenvolvimento de agentes inteligentes para o *7 Wonders*. Os trabalhos que antecedem este, buscaram obter o conhecimento de estratégias e o desenvolvimento de um ambiente que possibilitasse a construção e teste dos agentes. Estes foram os primeiros agentes inteligentes que foram desenvolvidos utilizando o ambiente e as estratégias até aqui conhecidas. O agente baseado em redes neurais, que foram treinadas de forma supervisionada, conseguiu aprender uma estratégia geral que se baseia na construção de cartas civis e militares. O agente desenvolvido com a técnica de *Deep Q-Learning* obteve um resultado semelhante, por outro lado também priorizou a construção de estruturas comerciais.

Como não foi possível testar os agentes contra humanos, os testes foram baseados em jogos contra um agente que joga de forma aleatória e partidas que os agentes desenvolvidos jogaram entre si. As pontuações médias dos agentes foram satisfatórias, visto que ficaram relativamente próximas da pontuação média dos melhores jogadores do BGA que é de 53 pontos.

Além disso, pode-se dizer que estes foram os primeiros passos para que sejam desenvolvidos agentes melhores e mais competitivos. Também vale considerar que a escolha da técnica a ser utilizada depende dos objetivos futuros, por exemplo, caso a pretensão seja construir um agente capaz de superar especialistas humanos, o *Deep Q-Learning* pode ser a técnica mais adequada. Já o aprendizado supervisionado se mostra útil quando queremos imitar o comportamento humano, fazendo com que redes neurais aprendam estratégias utilizadas por jogadores especialistas.

5.1 DIFICULDADES ENCONTRADAS

O *Board Game Arena*[1], que é a única plataforma online de jogos de tabuleiro que fornece uma implementação popular do *7 Wonders*, não possibilita o *download* ou disponibiliza tabelas relacionadas aos históricos das partidas. Por causa disso, as redes neurais treinadas de forma supervisionada, inicialmente, utilizaram um conjunto de dados relacionado ao histórico

[1] https://boardgamearena.com

de partidas do agente baseado em regras. Para no futuro seguir esta abordagem, se os termos e condições da plataforma permitirem, deve ser considerado o uso de *scripts* para automatizar a extração de dados a partir do *replay* das partidas.

Outra dificuldade encontrada durante o desenvolvimento deste trabalho foi a sincronização entre os agentes inteligentes e a implementação do *7 Wonders*. A comunicação por meio de arquivos, fez com que os códigos dos agentes tivessem muitas estruturas condicionais para garantir a sincronização. Isso tornou os códigos dos agentes, de certa forma, pouco intuitivos e gerou *bugs* que foram superados, mas foram difíceis de encontrar e resolver.

5.2 TRABALHOS FUTUROS

As estratégias que todos os agentes desenvolvidos utilizam acabam desconsiderando o gasto de moedas. Por exemplo, se o agente puder construir a carta *Panteão*, que fornece 7 pontos, gastando 10 moedas e também tiver a possibilidade de construir a carta *Senado*, que fornece 6 pontos, pelo custo de 2 moedas, considerando o peso atribuído a essas cartas, o agente irá sempre construir o *Panteão*. Neste caso, não há vantagem em construir o *Panteão*, pois há possibilidade de utilizar uma carta que fornece praticamente a mesma pontuação por um menor custo.

Além disso, as estratégias aprendidas pelos agentes não consideram o comércio com vizinhos, pois a implementação do *7 Wonders* faz a compra de recursos, quando necessária, de forma automática. Essa limitação pode levar a situações em que os agentes acabam dando moedas a vizinhos que anteriormente não tinham condições de comprar boas cartas, que poderiam ser utilizadas pelo agente. Isso acaba sendo uma vantagem para os adversários e ao mesmo tempo uma desvantagem para os agentes.

Tendo em vista as limitações discutidas anteriormente, as próximas evoluções dos 3 agentes desenvolvidos neste trabalho podem passar a também considerar outros aspectos do jogo, como as estratégias dos jogadores vizinhos, a escolha de qual vizinho comprar um determinado recurso e o custo das cartas. Também é interessante pensar em uma forma de testar as melhorias, e os agentes que possam vir a ser desenvolvidos, contra humanos.

Por fim, a técnica de *Deep Q-Learning*, tendo em vista seu potencial para este tipo de aplicação, pode ser melhor explorada. Neste trabalho foi desenvolvido um agente inicial, que pode ser melhorado. Isso pode ser feito através do teste de mais hiperparâmetros, melhorias no sistema de recompensas do agente e utilização de uma segunda rede neural chamada de *Target*, para que o treino ocorra de forma mais estável.

REFERÊNCIAS BIBLIOGRÁFICAS

ALPAYDIN, E. **Introduction to Machine Learning**. 3. ed. [S.l.]: The MIT Press, 2014. 613 p.

ASSUNÇÃO, J. et al. Data mining 7 wonders, the board game. In: SIMPÓSIO BRASILEIRO DE GAMES, 16., 2019, Rio de Janeiro. **Anais...** Sociedade Brasileira de Computação, 2019. Acesso em: 15/10/2020. Disponível em: <https://www.sbgames.org/sbgames2019/files/papers/ComputacaoShort/196631.pdf>.

BAUZA, A. **7 Wonders**: regras. [S.l.], 2010. 12 p. Acesso em 02 nov. 2020. Disponível em: <https://www.inboardgame.com.br/wp-content/uploads/7-Wonders-Livro-de-Regras-Manual.pdf>.

BRAGA, A. d. P.; LUDERMIR, T. B.; CARVALHO, A. P. d. L. F. **Redes neurais artificiais: teoria e aplicações**. [S.l.]: Livros Técnicos e Científicos, 2000. 248 p.

CAMPBELL, M.; JR, A. J. H.; HSU, F.-h. Deep blue. **Artificial intelligence**, Elsevier, v. 134, n. 1-2, p. 57–83, 2002.

DATA SCIENCE ACADEMY. **Deep Learning Book**. 2020. Acessado em 25 nov 2020. Disponível em: <http://deeplearningbook.com.br/aprendizado-com-a-descida-do-gradiente>.

FACELI, K. et al. **Inteligência Artificial**: Uma abordagem de aprendizado de máquina. Rio de Janeiro: Livros Técnicos e Científicos Editora Ltda., 2011. 375 p.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. 775 p.

HAYKIN, S. S. et al. **Neural networks and learning machines**. 3. ed. [S.l.]: New York: Prentice Hall, 2009. 906 p.

JARDIM, J. et al. An implementation of the 7 wonders board game for ai-based players. In: SIMPÓSIO BRASILEIRO DE GAMES, 17., 2020, Recife. **Anais...** Sociedade Brasileira de Computação, 2020. Acesso em: 20/11/2020. Disponível em: <https://www.sbgames.org/proceedings2020/ComputacaoShort/208553.pdf>.

LAMPLE, G.; CHAPLOT, D. S. Playing fps games with deep reinforcement learning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2017. v. 31, n. 1.

MNIH, V. et al. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013.

NG, A. Y.; JORDAN, M. I. **Shaping and policy search in reinforcement learning**. 2003. 143 p. Tese (Doutorado em Ciência da Computação) — University of California, Berkeley, Berkeley, 2003.

PLEINES, M. **Applying neural networks to the behavior of real-time strategy combat units in StarCraft**. 2016. 42 p. Monografia (Trabalho de Conclusão de Curso) — Universidade de Ciências Aplicadas do Reno-Waal, Cleves, 2016.

ROBILLIARD, D.; FONLUPT, C.; TEYTAUD, F. Monte-carlo tree search for the game of 7 wonders. In: SPRINGER. **Workshop on Computer Games**. [S.l.], 2014. p. 64–77.

RUSSELL, S. J.; NORVIG, P. **Inteligência artificial**: Tradução da terceira edição. 3. ed. Rio de Janeiro: Elsevier, 2013. 1016 p.

SCHAEFFER, J. et al. Checkers is solved. **science**, American Association for the Advancement of Science, v. 317, n. 5844, p. 1518–1522, 2007.

SILVER, D. et al. Mastering the game of go without human knowledge. **nature**, Nature Publishing Group, v. 550, n. 7676, p. 354–359, 2017.

STANESCU, M. et al. Evaluating real-time strategy game states using convolutional neural networks. In: IEEE. **2016 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.], 2016. p. 1–7.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

ŚWIECHOWSKI, M.; TAJMAJER, T.; JANUSZ, A. Improving hearthstone ai by combining mcts and supervised learning algorithms. In: IEEE. **2018 IEEE Conference on Computational Intelligence and Games (CIG)**. [S.l.], 2018. p. 1–8.

VINYALS, O. et al. Starcraft ii: A new challenge for reinforcement learning. **arXiv preprint arXiv:1708.04782**, p. 20, 2017.

XENOU, K.; CHALKIADAKIS, G.; AFANTENOS, S. Deep reinforcement learning in strategic board game environments. In: SPRINGER. **European Conference on Multi-Agent Systems**. [S.l.], 2018. p. 233–248.

YANNAKAKIS, G. N.; TOGELIUS, J. **Artificial intelligence and games**. [S.l.]: Springer, 2018. v. 2. 364 p.