UMA METODOLOGIA PARA O DESENVOLVIMENTO DE UM AGENTE JOGADOR DE *CONNECT-4*

A METHODOLOGY FOR THE DEVELOPMENT OF A *CONNECT-4* PLAYER AGENT

Pedro Michael Santos Soares*
Diego Rocha Lima**
Carina Teixeira de Oliveira***

RESUMO

Ao longo dos anos, muitos agentes inteligentes foram desenvolvidos para jogar contra humanos, principalmente jogos de tabuleiro. Nesses tipos de jogos, todas as jogadas possíveis são conhecidas, muito embora seja complexo para um humano mapear todas as jogadas dada a grande quantidade de configurações do tabuleiro. Assim, agentes cada vez mais inteligentes, dotados de algoritmos eficientes, fazem frente a jogadores reais. Neste trabalho, é apresentada uma metodologia composta por três etapas utilizadas para o desenvolvimento de um agente para o jogo *Connect-4*. A primeira etapa apresenta a modelagem do agente inteligente, com destaque para o uso do algoritmo *minimax* com *poda alfa-beta*. Em seguida, na segunda etapa, é detalhado o desenvolvimento do jogo *web* que foi integrado à modelagem do agente. Por fim, na última etapa é apresentada a realização da avaliação do agente e das métricas da aplicação. Ao final, são mostrados resultados referentes aos jogos realizados por 824 jogadores humanos contra o agente desenvolvido.

Palavras chaves: *Connect-4*. Agente Inteligente. Minimax. Poda alfa-beta.

ABSTRACT

Over the years, many intelligent agents have been developed to play against humans, especially board games. In these types of games, all possible moves are known, even though it is complex

* Graduando em Bacharelado em Ciência da Computação, Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: pedromtec@gmail.com
** Mestre em Ciência da Computação pela Universidade Federal Rural do Semi-Árido (UFERSA), Docente do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: diego.rocha@.ifce.edu.br
*** Doutora em Informática pela Université Joseph Fourier - UJF, Docente do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Aracati, Ceará, Brasil. E-mail: carina@lar.ifce.edu.br

for a human to map all moves given the large number of board configurations. Thus, increasingly intelligent agents, equipped with efficient algorithms, face real players. In this work, it is presented a methodology composed of three stages used for the development of an agent for the game *Connect-4*. The first stage presents a modeling of the intelligent agent, highlighting the use of the *minimax* algorithm with *alpha-beta pruning*. Then, in the second stage, the development of the web game that was integrated with the agent modeling is detailed. Finally, the last stage shows how an evaluation of the agent and application metrics was performed. In the end, it is presented the results related to games played by 824 human players against the developed agent.

Keywords: Connect-4. Intelligent Agent. Minimax. Alpha-beta Pruning.

1 INTRODUÇÃO

A Inteligência Artificial (IA) é um campo de estudo que vem crescendo bastante nos últimos anos. Acompanhando a velocidade do desenvolvimento tecnológico que temos, a tendência é que a área ganhe mais importância ainda com o passar do tempo (YANNAKAKIS; TOGELIUS, 2018). A IA engloba uma ampla variedade de problemas e subáreas de estudo, cada uma dentro da sua especificidade. Isso faz com que ela traga contribuições nas mais diferentes áreas do conhecimento, inclusive jogos (NORVIG; RUSSELL, 2014).

Um dos primeiros domínios na pesquisa em IA foi desenvolver um adversário para o jogo de xadrez de computador (ALLIS, 1988), algo que não surpreende, considerando-se as possibilidades que os pesquisadores pensavam que os computadores poderiam alcançar em um futuro próximo. Usando essas possibilidades, os pesquisadores pensaram que seria possível deixar o computador realizar tarefas para as quais os humanos precisam de inteligência, sejam elas quais forem (ALLIS, 1988).

Com isso, a área de jogos, principalmente de jogos de tabuleiro, vem sendo um dos principais campos de domínio para tentativas de IA, pois são ambientes formais e altamente restritos para tomadas de decisões (YANNAKAKIS; TOGELIUS, 2018). O uso de IA em jogos de tabuleiro ganhou bastante popularidade em 1996, quando o programa de computador *Deep Blue* conseguiu derrotar o campeão mundial de xadrez Garry Kasparov em uma partida de xadrez (CAMPBELL; JR; HSU, 2002).

Outro marco importante aconteceu no Jogo *Go*. Esse jogo há muito tempo é visto como um dos mais desafiadores dos jogos clássicos para IA devido ao seu enorme espaço de busca e à dificuldade de avaliar as posições e movimentos do tabuleiro. O programa de computador *AlphaGo*, desenvolvido pelo *Google DeepMind*, venceu uma partida de cinco jogos contra *Lee Sedol*, um profissional com 9 anos de experiência e campeão mundial do jogo na época (GIBNEY, 2016).

Um outro jogo de tabuleiro bastante explorado na literatura de IA é o *Connect-4*, um jogo simples no qual dois jogadores se alternam jogando discos em um tabuleiro 7×6. O primeiro jogador a obter quatro discos em sequência (vertical, horizontal ou diagonal) vence. O jogo ficou

inicialmente conhecido como "*The Captain's Mistress*", mas foi lançado em sua forma atual por Milton Bradley em 1974 (ALLIS, 1988).

O *Connect-4* foi resolvido em (ALLIS, 1988). Nesse trabalho, Victor Allis introduziu uma abordagem baseada em conhecimento e várias estratégias vencedoras. Inclusive, Allis também demonstrou que com um jogo perfeito de ambos os jogadores, o primeiro jogador sempre pode vencer se jogar na coluna do meio primeiro. Além disso, se o primeiro jogador escolher outra coluna, o segundo jogador sempre pode forçar pelo menos um empate.

A complexidade do *Connect-4* é considerada alta (ALLIS, 1988). Cada posição do tabuleiro pode estar em um de três possíveis estados: vazio, amarelo ou azul. Portanto, o número de configurações possíveis do tabuleiro é no máximo $3^{42} \geq 10^{20}$. Este limite superior é muito bruto e pode ser levado a melhores proporções por várias configurações serem irregulares. Por exemplo, se uma posição é vazia em uma coluna, então todas as posições acima dela também devem ser vazias. Remover essas configurações inválidas proporciona um limite superior melhor de $7,1 \times 10^{13}$ possíveis estados válidos (ALLIS, 1988).

Apesar das estratégias introduzidas no trabalho de (ALLIS, 1988), formas de otimizar os algoritmos da pesquisa não foram levadas em consideração. Em um cenário real, o tempo de resposta de um algoritmo é essencial, pois nenhum jogador ficaria satisfeito em esperar minutos ou até horas pela jogada de seu adversário.

Um dos algoritmos clássicos aplicados em jogos de dois ou mais jogadores com objetivos conflitantes é chamado *minimax* (KORF, 1999). É um tipo de algoritmo de busca usado na tomada de decisões e na teoria dos jogos para encontrar o movimento ótimo para um jogador, assumindo que seu oponente também joga de forma ótima. É amplamente utilizado em jogos baseados em turnos para dois jogadores, como Jogo da Velha, Gamão, Dama, Xadrez, entre outros (NORVIG; RUSSELL, 2014).

O algoritmo *minimax* é bom em prever o movimento do oponente e, então, vencê-lo. Porém, o tempo de execução do *minimax* é sempre um problema quando o espaço de busca do problema é grande. A fim de diminuir o tempo de execução do *minimax*, estratégias como a *poda alfa-beta* podem ser utilizadas. O tempo para o *minimax* analisar cada nó na árvore do jogo pode ser demasiadamente alto. Dessa forma, o objetivo principal da *poda alfa-beta* é calcular a decisão correta sem examinar todos os nós da árvore, podando ramificações que não influenciam a decisão final (NORVIG; RUSSELL, 2014).

No entanto, mesmo que o tempo de execução seja reduzido, em alguns casos o tempo para o *minimax* atingir níveis de otimização de tempo é considerável. Por exemplo, dependendo do estado do tabuleiro, o *minimax* não tem a capacidade de pesquisar o valor ótimo em um nível muito baixo da árvore do jogo em uma velocidade aceitável. A fim de acelerar o progresso de fazer o melhor movimento, funções heurísticas podem ser aplicadas.

As funções heurísticas determinam qual ramo seguir, classificando as alternativas em cada *branch-step* com base nas informações disponíveis. Uma vez que o *minimax* escolhe o mais alto valor que a heurística gera, a heurística se torna tão essencial que qualquer mudança nela pode alterar o resultado final do movimento.

Neste trabalho, é apresentada a metodologia utilizada para o desenvolvimento de um agente capaz de jogar *Connect-4*. O objetivo principal é mostrar como desenvolver uma IA capaz de jogar *Connect-4* e colocar o jogo em produção para vários usuários jogarem. O algoritmo *minimax* é utilizado junto com uma estratégia *poda alpha-beta* e uma heurística para avaliar os estados do tabuleiro. Dentro das etapas da metodologia proposta, é apresentada uma versão *web* do jogo desenvolvido, que foi disponibilizado para usuários de todo o mundo acessarem e jogarem. Pode-se dizer que o jogo ficou facilmente acessível, pois a versão disponibilizada se ajusta facilmente a qualquer dispositivo, facilitando a jogabilidade e usabilidade de usuários que preferem jogar pelo celular, *tablet* ou computador.

O restante do artigo está organizado da seguinte forma. Na Seção 2 são apresentados os principais conceitos necessários para a compreensão da proposta do trabalho. Na Seção 3 são mostrados os trabalhos com ideias semelhantes à pesquisa corrente. A Seção 4 detalha a metodologia da proposta. Na Seção 5 são mostradas as métricas do agente, tais como: número de vitórias, derrotas e empates. Métricas relacionadas ao público alvo também são exibidas. Finalmente, na Seção 6 o trabalho é concluído e apresenta direcionamentos para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

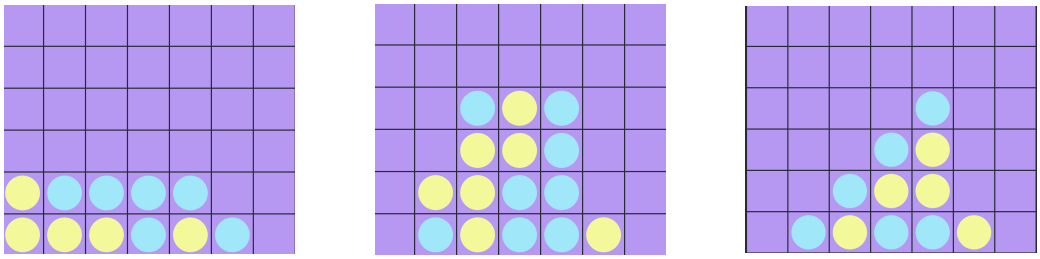
Nesta seção, são apresentados os principais conceitos necessários para a compreensão da proposta do trabalho, como as regras do jogo, objetivos, organização em árvore e particularidades dos algoritmos que são utilizados.

2.1 Regras do jogo *Connect-4*

O *Connect-4* é um jogo de tabuleiro para dois jogadores. Como ilustrado na Figura 1, o tabuleiro é dividido em uma grade de 6 linhas e 7 colunas. No início da partida, cada jogador recebe 21 peças, representando dois conjuntos. Cada conjunto tem peças de cores diferentes. Neste trabalho, os conjuntos são representados pelas cores azul e amarela.

O objetivo do jogo é conectar quatro peças da mesma cor na horizontal, vertical ou diagonal, conforme exemplificam as Figuras 1a, 1b e 1c, respectivamente, nas quais as peças azuis vencem o jogo. A dinâmica do jogo consiste em cada jogador fazer um único movimento por turno, colocando uma de suas peças em uma das 7 colunas da grade. A peça vai cair em linha reta na coluna escolhida, ocupando a posição não ocupada mais baixa daquela coluna. Nota-se que assim que uma coluna contiver 6 peças, nenhuma outra peça poderá ser colocada nela. Nesse jogo as peças não podem ser movimentadas após entrarem no tabuleiro, assim um "movimento"pode ser entendido apenas como o ato de inserir uma nova peça no tabuleiro. O jogador que chegar ao objetivo primeiro, vence o jogo. Se todas as 42 peças foram jogadas e nenhum jogador conseguir conectar 4 de suas peças em linha, então a partida é finalizada sem vencedores, caracterizando um empate.

Figura 1 – Exemplos de tabuleiro de *Connect-4* com peças azuis vencedoras.



(a) Azuis vencem na horizontal (b) Azuis vencem na vertical (c) Azuis vencem na diagonal

Fonte: Elaborada pelos autores (2021).

2.2 Árvore de jogo

Uma árvore de jogo pode ser entendida como uma estrutura de dados hierárquica, em forma de árvore e consistindo de todos os possíveis movimentos que levam o jogador de um estado para outro. Nesta árvore, os nós representam os estados do tabuleiro e as arestas indicam como um estado pode ser transformado em outro através de um movimento (NORVIG; RUSSELL, 2014). Formalmente, é possível representar um jogo como um problema de busca com os seguintes componentes:

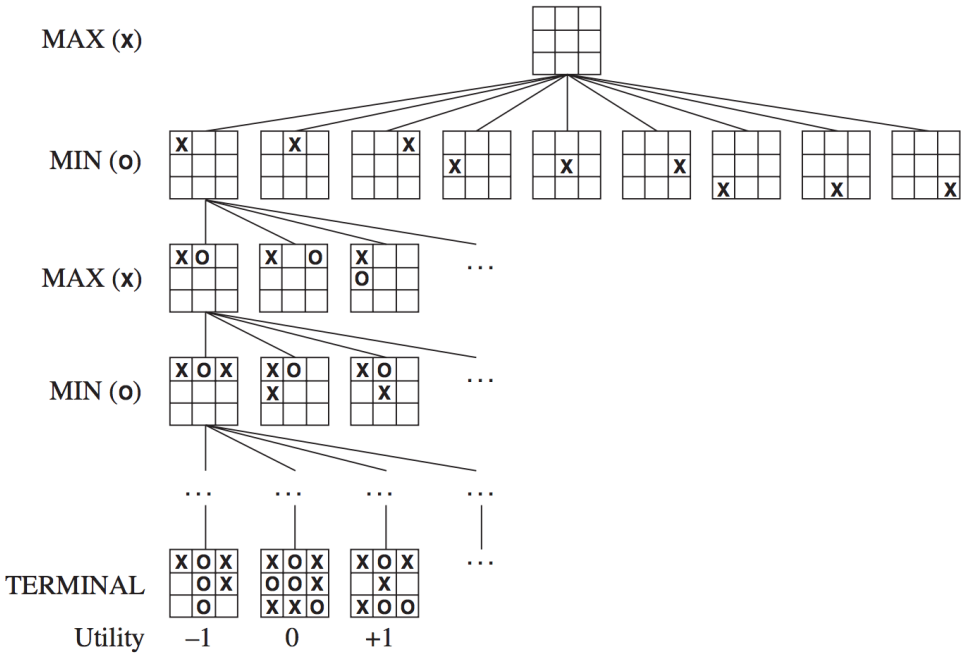
- **Estado** S_0: Representa o **estado inicial** do jogo.
- **Conjunto de Estados**: Representa todos os estados válidos do jogo.
- **Ações**: Define quais são os movimentos válidos de um estado S.
- **Função de teste de término** T:

$$T(S) = \begin{cases} 1, & \text{se S é um estado terminal.} \\ 0, & \text{se S não é um estado terminal.} \end{cases}$$

- **Utilidade**(S): Define um valor numérico para um estado terminal S. Por exemplo, no *Jogo da Velha*, o resultado pode ser uma vitória, derrota ou um empate do primeiro jogador, e estes podem ser representados pelos valores +1, -1 ou 0, respectivamente. Uma função utilidade também pode ser chamada de **função objetivo** ou **função compensação**.

A Figura 2 ilustra uma visão parcial de uma árvore para o Jogo da Velha. Nota-se que, no **estado inicial**, *MAX* tem nove **ações** possíveis. O jogo se alterna entre a colocação de um *X* por *MAX* e um *O* por *MIN* até que se alcance nós folhas que caracterizam os **estados finais** do jogo. Cada nó folha possui um valor associado que representa sua **utilidade**, na qual valores altos são considerados bons para o jogador *MAX* e ruins para *MIN*.

Figura 2 – Visão parcial da Árvore de Busca para o Jogo da Velha.



Fonte: (NORVIG; RUSSELL, 2014).

2.3 Algoritmo *Minimax*

O algoritmo *minimax* é um técnica que visa encontrar um movimento ótimo em um jogo com dois jogadores de desempenho ótimo. Em uma árvore de busca para dois jogadores, existem dois tipos de nós: os nós que representam os movimentos do agente *MAX* e os nós que representam os movimentos do oponente *MIN* (NORVIG; RUSSELL, 2014). Quando está em um nível *MAX*, o algoritmo segue a aresta que tem como destino o nó com pontuação mais alta. Quando está em um nível *MIN*, o algoritmo escolhe um movimento que tem como destino o nó com pontuação mais baixa. A pontuação (ou valor *minimax*) de um nó pode ser calculada a partir do procedimento recursivo:

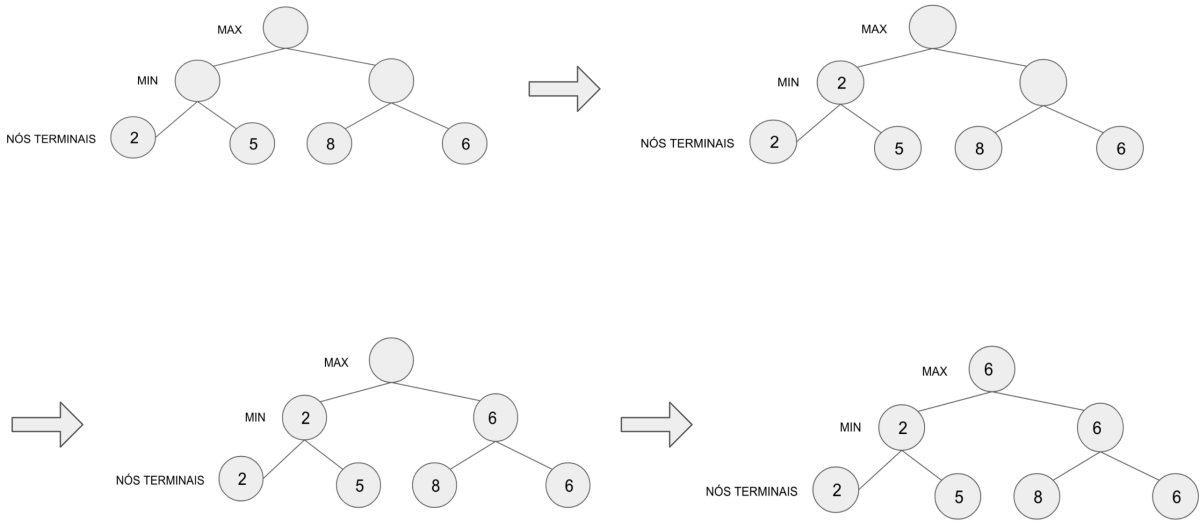
$$minimax(n) = \begin{cases} utilidade(n), & \text{se } n \text{ é um nó terminal} \\ max(minimax(s_0)\ldots minimax(s_k)) \quad \forall s \in sucessores, & \text{se } n \text{ é um nó MAX} \\ min(minimax(s_0)\ldots minimax(s_k)) \quad \forall s \in sucessores, & \text{se } n \text{ é um nó MIN} \end{cases} \tag{1}$$

onde n é o nó atual, *sucessores* o conjunto de nós que podem ser gerados a partir de n e *max* é uma função que retorna o maior valor dentre um conjunto de valores. O algoritmo *minimax* executa uma exploração completa da árvore de jogo fazendo uma busca em profundidade. Se a profundidade máxima da árvore é m e existem b movimentos válidos em cada nó, a complexidade de tempo do algoritmo é $O(b^m)$ (NORVIG; RUSSELL, 2014).

Os exemplos a seguir mostram como o *minimax* se comporta em uma árvore de jogo de duas jogadas. A Figura 3 exemplifica o funcionamento do algoritmo. No segundo nível da árvore observa-se um movimento de *MIN*, sendo replicado os valores 2 e 6 respectivamente que são

os menores valores advindos do nível 3. O primeiro nível da árvore é um movimento de MAX, onde é replicado o valor 6, que é o maior valor advindo do nível 2.

Figura 3 – Exemplo do algoritmo *minimax* em uma árvore de jogo de duas jogadas.



Fonte: Elaborada pelos autores (2021).

Como visto, o algoritmo *minimax* sempre encontra o movimento ótimo, podendo mapear todas as possibilidades do jogo que levam a uma vitória. Porém sua complexidade assintótica é relativamente alta, já que o algoritmo examina todos os possíveis estados a partir do estado atual. O custo dessa abordagem é inviável para jogos como o *Connect-4*, que tem aproximadamente $7.1 * 10^{13}$ possíveis estados válidos (ALLIS, 1988). Isso nos leva a adicionar estratégias eficientes que podem ser utilizadas juntamente ao algoritmo *minimax*.

2.4 *Poda alfa-beta* e heurísticas

A primeira otimização que pode ser feita e foi implementada no algoritmo *minimax* é a utilização da *poda alfa-beta*. Uma primeira característica que deve ser observada é que a *poda alfa-beta* não altera o resultado do *minimax*, ela é um aperfeiçoamento do algoritmo, que corresponde ao cálculo da decisão correta sem navegar por todos os nós da árvore, podando ramificações que comprovadamente não levam a um resultado melhor que o atual (NORVIG; RUSSELL, 2014).

O algoritmo mantém duas novas informações: os parâmetros *alpha* e *beta*. O parâmetro *alpha* representa o melhor valor (mais alto), até o momento, escolhido por *MAX*. Já o parâmetro *beta* representa o melhor valor (mais baixo), até o momento, escolhido por *MIN*. Inicialmente, os dois parâmetros iniciam com os piores valores possíveis: *alpha* recebe o valor “infinito negativo” e *beta* o valor “infinito positivo”.

Para exemplificar o funcionamento do algoritmo, a sua execução é dividida em dois cenários:

- **Nó no nível** *MIN*: Caso a pontuação mínima que é garantida pelo jogador *MIN* se torne menor do que a pontuação máxima que é garantida ao jogador *MAX* ($beta \leq alpha$), o "nó pai" não deve selecionar esse nó, pois isso fará com que a pontuação para o nó pai seja pior. Com isso, os outros ramos do nó não precisarão ser explorados.

- **Nó no nível** *MAX*: Caso a pontuação máxima que é garantida pelo jogador *MAX* se torne maior do que a pontuação mínima que é garantida ao jogador *MIN* ($alpha \geq beta$), o "nó pai" não deve selecionar esse nó, pois isso fará com que a pontuação para o nó pai seja pior. Com isso, os outros ramos do nó não precisarão ser explorados.

A efetividade da *poda alfa-beta* depende altamente de como os nós sucessores são explorados. Caso os nós mais promissores sejam examinados primeiro, apenas $O(b^{m/2})$ nós precisam ser examinados, em vez de $O(b^m)$ para o algoritmo minimax. Se os sucessores forem examinados de forma aleatória, o número de nós examinados será cerca de $O(b^{3m/4})$ (NORVIG; RUSSELL, 2014).

Apesar da *poda alfa-beta* melhorar o desempenho do algoritmo, ela ainda tem que chegar nos estados terminais, sendo ineficiente para jogos com muitos passos até os estados terminais. Para resolver esse problema, foi necessário substituir a função de utilidade por uma função de avaliação heurística e substituir o teste de término por um teste de corte. O teste de corte foi usado para definir o nível máximo de profundidade que o algoritmo deve percorrer, já a função heurística vai retornar uma estimativa de utilidade esperada do jogo a partir de um dado estado.

Nos estados não terminais, a função de avaliação deve estar fortemente correlacionada com as chances reais de vitória, calculando as características de um determinado estado. Tais características podem ser agrupadas por uma função linear ponderada:

$$AVAL(S) = w_1 f_1(S) + w_2 f_2(S) + ... + w_n f_n(S) = \sum_{i=1}^{n} w_i f_i(S) \tag{2}$$

No Xadrez, $f_i(S)$ poderia representar o número de peças i (peão, bispo, etc) em um dado estado, w_i seria o peso da característica i, por exemplo, peão teria o peso 1, cavalo (3), bispo (3), torre (5), rainha (9). A seguir é apresentado um código em Python com aplicação da *poda alfa-beta* e heurísticas:

```
import math
def alphabeta(node, depth, alpha, beta, maximizingPlayer):
  if depth == 0 or isTerminal(node):
    return heuristic(node)
  if maximizingPlayer:
    value = -math.inf
    for child in node:
      value = max(value, alphabeta(child, depth-1, alpha, beta, False))
      alpha = max(alpha, value)
      if alpha >= beta:
```

```
11          break
12        return value
13    else:
14      value = math.inf
15      for child in node:
16        value = min(value, alphabeta(child, depth-1, alpha, beta, True))
17        beta = min(beta, value)
18        if beta <= alpha:
19          break
20        return value
```

A função acima recebe como parâmetro o estado atual (*node*), a limitação de profundidade da busca (*depth*), os parâmetros *alpha* e *beta* e por um último um valor booleano (*maximizingPlayer*) que diz se o jogador atual é o jogador *MAX*. Na linha 3 é checado se o limite de profundidade foi atingido ou se o *node* atual representa um estado terminal. Nesse caso, o valor da heurística do *node* é retornado. Na linha 5 é verificado se o jogador é um jogador maximizador, caso a condição seja verdadeira, o valor minimax é calculado para *MAX* e o parâmetro alpha atualizado, caso contrário, o valor minimax é calculado para *MIN* e o parâmetro beta atualizado.

3 TRABALHOS RELACIONADOS

Existem vários trabalhos com estratégias de IA para jogos de tabuleiro. Essa seção aborda os principais marcos da IA em jogos de tabuleiro, além de trabalhos cujo algoritmo *minimax* foi aplicado no *Connect-4*.

Um dos projetos que ganhou repercussão mundial na área de IA foi a máquina de jogar xadrez *Deep Blue*, a qual derrotou o então campeão mundial de xadrez Garry Kasparov em uma partida de seis jogos em 1997. Uma imagem desse momento marcante pode ser vista na figura 4. O *Deep Blue* foi a combinação de um supercomputador e um software, ambos criados pela empresa IBM (CAMPBELL; JR; HSU, 2002). A solução foi baseada em uma estratégia de busca *minimax*, e contou com vários fatores para o sucesso. Dentre os quais podemos citar alguns. Um mecanismo de busca de xadrez de chip único. Um sistema maciçamente paralelo com vários níveis de paralelismo. Uma forte ênfase nas extensões de pesquisa. Uma complexa função heurística. E ainda o uso eficaz de um banco de dados com 700.000 partidas de xadrez.

Outro marco importante aconteceu no Jogo *Go*, que é um jogo estratégico para tabuleiro onde dois jogadores posicionam alternadamente pedras pretas e brancas, com objetivo de cercar a maior quantidade possível de território com suas próprias pedras. Esse jogo há muito tempo é visto como o mais desafiador dos jogos clássicos para IA devido ao seu enorme espaço de busca e à dificuldade de avaliar as posições e movimentos do tabuleiro, como visto na figura 5. Desde 2012, o *Google Deep Mind* vem elaborando pesquisas no contexto do jogo.

O primeiro trabalho elaborado pelo *Google Deep Mind* foi o *AlphaGo*, um programa de computador que combina árvore de pesquisa avançada com redes neurais profundas. Essas

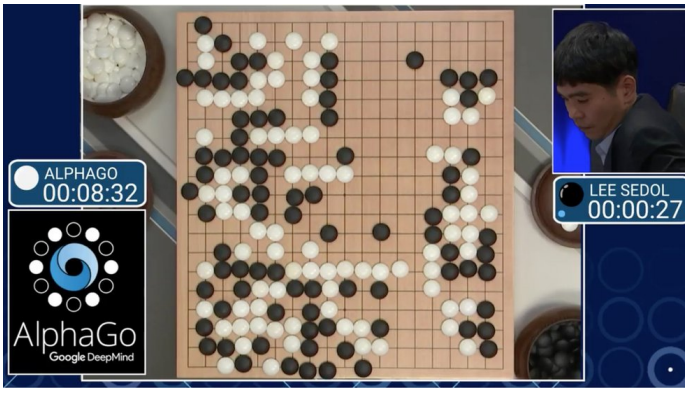
Figura 4 – Garry Kasparov vs. Deep Blue

Fonte: Stanford (2013).

redes neurais recebem como entrada o estado do tabuleiro e o processam por meio de várias camadas de rede diferentes, contendo milhões de conexões semelhantes a neurônios (SILVER et al., 2016).

Uma das redes neurais, a “rede de políticas”, seleciona o próximo movimento a ser executado. A outra rede neural, a “rede de valor”, prevê o vencedor do jogo. O *AlphaGo* foi treinado com vários jogos amadores para ajudá-lo a desenvolver uma compreensão do jogo humano razoável. Após isso, ele jogou contra diferentes versões de si mesmo milhares de vezes, cada vez aprendendo com seus erros. O *AlphaGo* derrotou o campeão europeu do *Go* humano por 5 games a 0 (SILVER et al., 2016), logo depois venceu uma disputa de cinco jogos contra *Lee Sedol*, campeão mundial de *Go* na época (GIBNEY, 2016).

Figura 5 – AlphaGo vs. Lee Sedol



Fonte: BBC (2016).

Um dos primeiros trabalhos sobre o *Connect-4* foi desenvolvido por Victor Allis em 16 de outubro de 1988, (ALLIS, 1988). Em seu trabalho, Allis descreve uma abordagem baseada em conhecimento, com nove estratégias, como uma solução para o *Connect-4*. Allis também descreve estratégias vencedoras em sua análise do jogo. Na época, a análise de força bruta não era considerada viável devido à complexidade do jogo e à tecnologia de computador

disponível. Devido às limitações, otimizações de tempo de processamento das estratégias não foram consideradas.

Em complemento ao trabalho de Victor Allis, que focava em uma abordagem baseada em conhecimento e estratégias vencedoras, o trabalho (KANG; WANG; HU, 2019) focou no desenvolvimento e análise de heurísticas. Neste trabalho, foram apresentadas heurísticas auto-desenvolvidas a partir de resultados bem demonstrados de pesquisas anteriores e experiências de partidas disputadas contra versões disponíveis do *Connect-4*. O trabalho conduziu três experimentos sobre a relação entre funcionalidade, profundidade da busca *minimax* e o número de características das heurísticas. O trabalho descobriu que conforme a profundidade da busca aumenta, a heurística tem melhores resultados. Além disso, mostrou-se que conforme o número de características aumenta, uma heurística se torna mais eficaz. Outro fator descoberto é que aumentando a profundidade de busca de uma heurística mais fraca, com menos características, essa método "mais fraco" pode vencer uma heurística com mais características (KANG; WANG; HU, 2019).

No trabalho (MISSURA; GAERTNER, 2008) foi desenvolvido um agente *adaptativo online* para o *Connect-4*. Nesse caso, *adaptativo* significa que o agente é capaz de modificar sua estratégia de jogo dependendo das habilidades de seu oponente e *online* significa que o raciocínio do agente é baseado nos eventos acontecendo apenas no jogo atual. O problema principal que esse trabalho busca resolver é escolher o nível ideal do agente em função do adversário. O agente era capaz de escalar sua dificuldade em função do jogador atual. O desempenho do agente desenvolvido foi avaliado através de jogos contra oponentes algorítmicos e humanos.

4 PROPOSTA

Nessa seção são apresentadas as três etapas da metodologia utilizada para desenvolvimento da proposta. A primeira etapa aplica os conceitos abordados no referencial teórico para a modelagem do agente. A segunda etapa apresenta o desenvolvimento e arquitetura do jogo. Por fim, na terceira etapa é mostrada como foi realizada a avaliação dos agentes e métricas de avaliação.

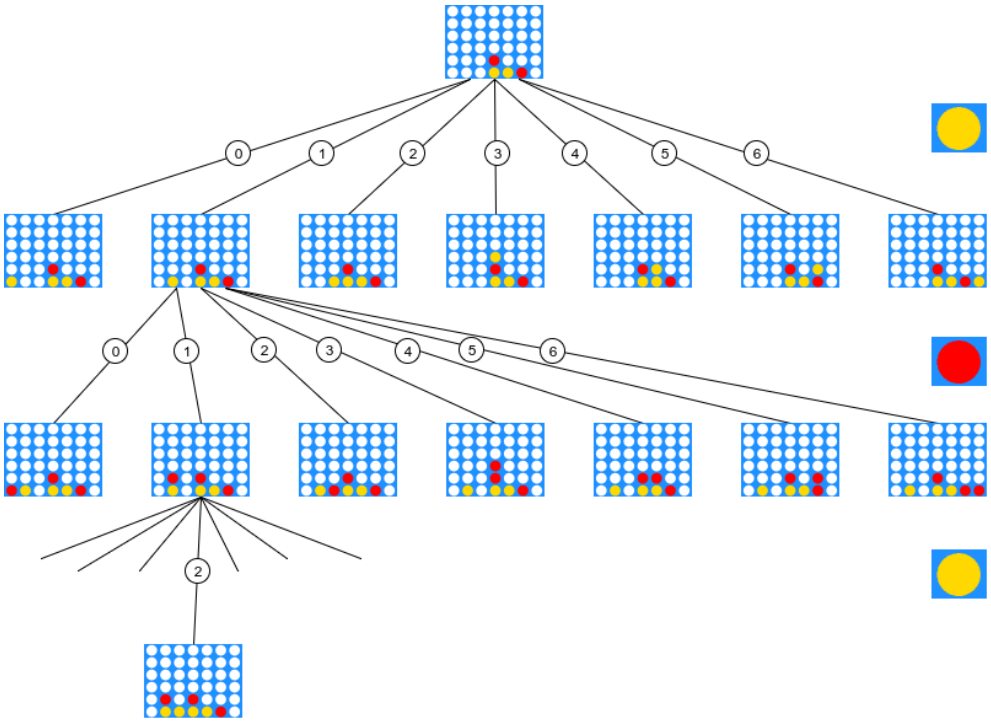
4.1 Etapa 1 - Modelagem do Agente Inteligente

É possível representar o *Connect-4* como um problema de busca, onde o estado é representado por uma determinada configuração do tabuleiro. No início do jogo, o primeiro jogador pode escolher uma coluna entre sete para colocar o disco colorido. Existem 7 colunas no total, portanto, existem 7 ramos para prosseguir na árvore. O segundo pode escolher uma coluna entre sete, continuando a partir da escolha do primeiro jogador.

Existem alguns casos especiais na árvore de jogo. Primeiro, se ambos os jogadores escolherem a mesma coluna 6 vezes no total, essa coluna não estará mais disponível para nenhum dos jogadores. Isso significa que o número de possibilidades de escolhas é reduzido em um para ambos os jogadores. Em segundo lugar, quando os dois jogadores fazem todas as escolhas

possíveis e ainda não há 4 discos em uma fileira, o jogo termina empatado. Finalmente, se qualquer jogador fizer 4 em sequência, o jogo termina. A Figura 6 mostra a representação parcial de uma árvore para o *Connect-4*.

Figura 6 – Visão parcial da Árvore de Busca para o *Connect-4*.



Fonte: Towards Data Science[1]

Esse tipo de estrutura faz com que seja possível a aplicação do algoritmo *minimax* junto com a *poda alfa-beta*. Mas ainda assim é necessário o uso de uma heurística para que seja possível encontrar boas jogadas em tempo aceitável, pois o agente foi testado contra usuários em partidas reais.

A heurística desenvolvida é simples e de fácil implementação. Ela se baseia em aplicar pesos para as características de um determinado estado do tabuleiro. Supondo que o agente joga com os discos amarelos, os pesos podem ser distribuídos conforme mostrado na tabela 1.

Tabela 1 – Heurística para o *Connect-4*.

Heurística	
Característica	**Peso**
Disco amarelo no centro (1)	10
Fileira com 2 discos amarelos e 0 discos vermelhos (2)	4
Fileira com 3 discos amarelos e 0 discos vermelhos (3)	10
Fileira com 0 discos amarelos e 3 discos vermelhos (4)	-10
Fileira com 4 discos amarelos (5)	100

As pontuações mostradas na tabela serão aplicadas a todas as possíveis fileiras de quatro células de um determinado estado do tabuleiro e o somatório dessas pontuações será retornado

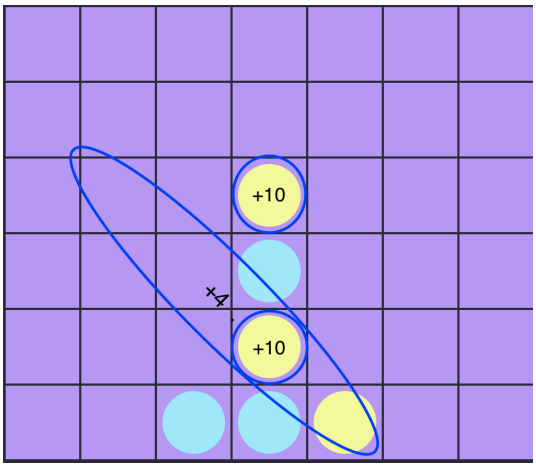
[1] https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0

pela função heurística. Formalmente, $f_i(S)$ retorna a quantidade de vezes que a característica i acontece no estado S e w_i retorna o peso da característica i, acarretando na seguinte função linear ponderada:

$$AVAL(S) = w_1 f_1(S) + w_2 f_2(S) + ... + w_5 f_5(S) = \sum_{i=1}^{5} w_i f_5(S) \quad (3)$$

A complexidade da heurística é de $O(n^m)$, onde n é o número de linhas e m é o número de colunas do tabuleiro. A Figura 7 ilustra como a heurística calcularia um determinado estado do tabuleiro. No exemplo, o estado possui dois discos amarelos no centro e uma fileira com 2 discos amarelos e 0 discos azuis, resultando no valor final $10 \times 2 + 4 \times 1 = 24$ como resultado da heurística.

Figura 7 – Exemplo de aplicação da heurística.



Fonte: Elaborada pelos autores (2021).

4.2 Etapa 2 - Desenvolvimento do jogo na web integrado ao algoritmo

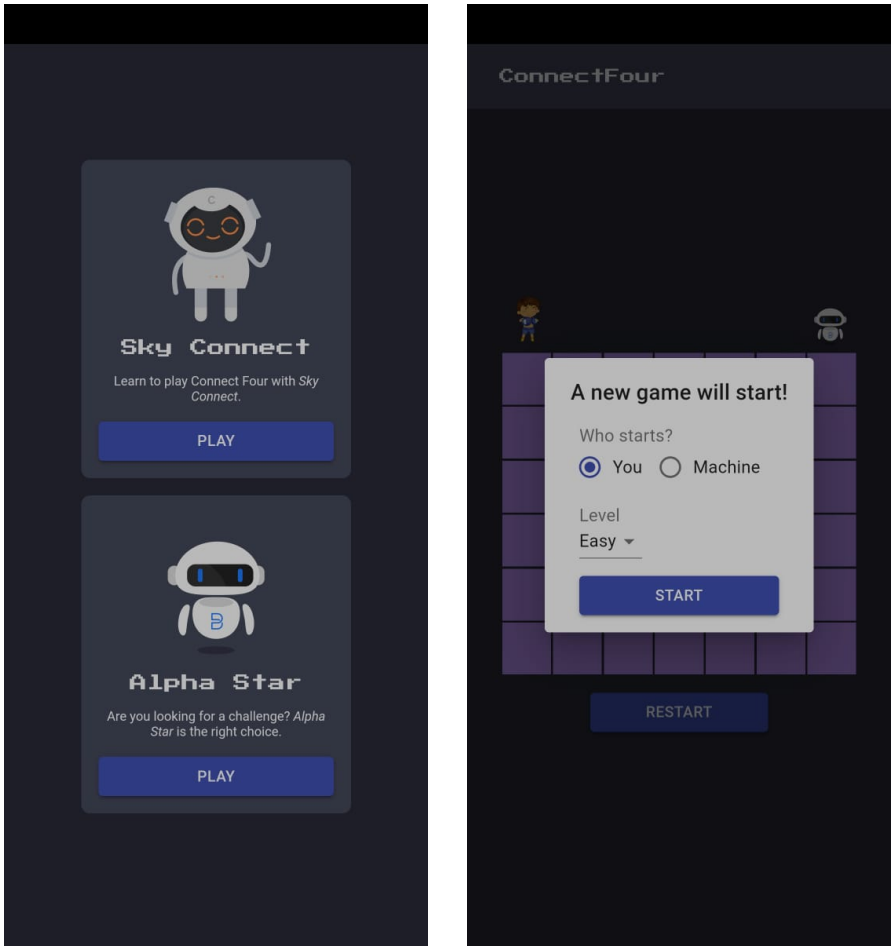
Inicialmente, o desenvolvimento do jogo foi dividido em *Frontend* e *Backend*. A aplicação *Frontend* é responsável por toda parte visual e regras do jogo. Ela foi desenvolvida utilizando a biblioteca *React* (REACT, 2021), que é uma biblioteca para criar interfaces de usuário usando o paradigma declarativo. O *Frontend* é dividido em duas partes, na primeira parte o usuário escolhe contra qual agente gostaria de jogar. Na segunda parte, o usuário pode escolher o nível do agente e iniciar a partida. As Figuras 8a e 8b ilustram como o *Frontend* exibe essas duas partes.

Como mostra a tela da Figura 8, existem dois agentes na aplicação *Frontend*, um chamado *Sky Connect* e outro chamado *Alpha Star*. Os níveis *easy*, *medium* e *hard* do *Sky Connect* representam o *minimax* com as limitações de profundidade 3, 4 e 5, respectivamente. Já os níveis *easy*, *medium* e *hard* do *Alpha Star* representam as limitações de profundidade 5, 6 e 7, respectivamente.

O tabuleiro do jogo é representado por uma matriz 6 x 7 de números inteiros, no qual o valor 0 representa uma célula vazia, 1 uma célula com uma peça do jogador e 2 uma célula com uma peça do agente. A Figura 9 exemplifica como um estado do tabuleiro pode ser representado

por uma matriz de inteiros. Na imagem da esquerda é possível observar como os dados são armazenados na matriz, já a da direita mostra como a matriz é renderizada para o usuário final. Além do tabuleiro, outras informações são armazenadas no *Frontend*, tais como o jogador atual da rodada e o nível de dificuldade do agente escolhido.

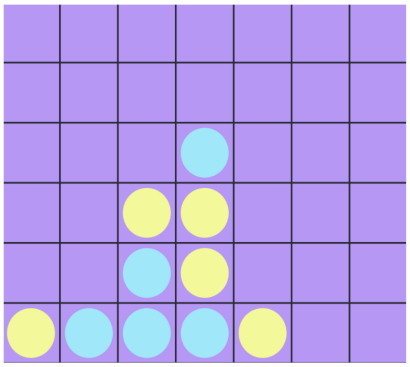
Figura 8 – Telas do jogo.



(a) Tela Inicial (b) Escolha de Nível

Figura 9 – Exemplo de matriz do estado do tabuleiro e sua representação renderizada.

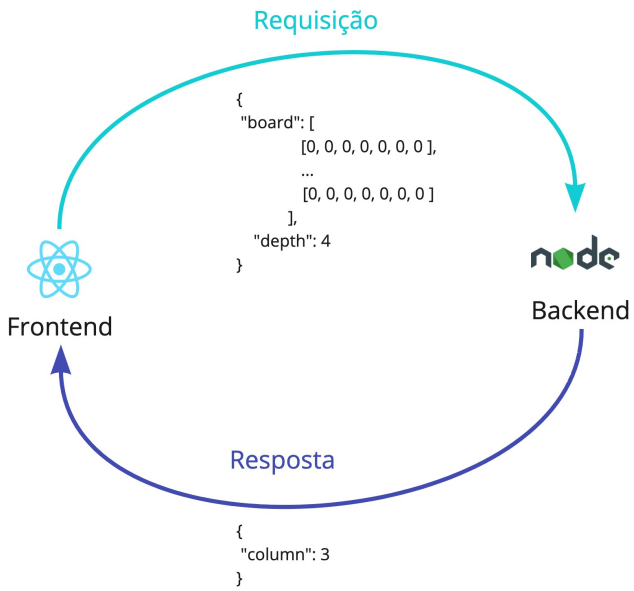
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	2	2	0	0	0
0	0	1	2	0	0	0
2	1	1	1	2	0	0



Fonte: Elaborada pelos autores (2021).

O *Backend* da aplicação guarda a lógica do algoritmo *minimax*. Como mostra a Figura 10, um exemplo de requisição entre cliente e servidor, o *Backend* é uma aplicação *Node.js* (NODE.JS, 2021) que provê um *endpoint* único que recebe o estado do tabuleiro e um parâmetro de limitação de profundidade no corpo da requisição. Quando os dados chegam, são processados pelo algoritmo *minimax*, gerando como resposta a coluna que deve ser escolhida pelo agente.

Figura 10 – Comunicação entre cliente e servidor.



Fonte: Elaborada pelos autores (2021).

4.3 Etapa 3 - Avaliação dos agentes e métricas da aplicação

A última etapa foi a avaliação dos agentes e das métricas da aplicação. Foram armazenadas as informações de cada partida, tais como o agente selecionado, o nível do agente e o resultado final da partida. Essas informações foram armazenadas através do serviço *Firestore* da plataforma *Firebase* (FIREBASE, 2021). O *Firebase* é uma plataforma do Google que fornece ferramentas para desenvolver aplicações de alta qualidade em um curto intervalo de tempo. O fato de ser um caso de uso onde era necessário salvar apenas os resultados das partidas fez com o que o serviço *Firestore* se tornasse ideal.

Para coletar métricas dos usuários foi utilizado o *Google Analytics* (ANALYTICS, 2021), uma ferramenta para monitoramento de desempenho de um site. Oferecida gratuitamente pela *Google*, a ferramenta permite mensurar o número de visitantes do site. Além disso, foi possível saber através de qual dispositivo os usuários mais acessaram a aplicação e até mesmo a localização geográfica dos mesmos.

A etapa de avaliação foi realizada em um período de 3 meses que aconteceu entre 1 de Setembro de 2020 a 30 de novembro de 2020. O link *https://connectfour-ai.vercel.app/* do jogo foi divulgado através de grupos do eixo de computação e por meio das redes sociais.

Na próxima seção, são apresentados os principais resultados obtidos nesta etapa de avaliação.

5 RESULTADOS

Nesta seção de resultados, primeiramente, são ilustradas as métricas da aplicação que foram obtidas através do *Google Analytics*. Em seguida, são exibidos as informações que foram coletadas a partir da plataforma *Firebase*, ilustrando os resultados dos agentes em diferentes níveis de dificuldade.

A Tabela 2 apresenta a distribuição de usuários por país. Durante o período de 3 meses, foram registrados 824 usuários. O Brasil foi o país com mais usuários. Porém, apesar de uma ampla divulgação local, o jogo também teve um alcance de nível global, acarretando em partidas contra usuários de outros 12 países de diferentes partes do mundo, com países da América do Sul, América do Norte e Europa.

Tabela 2 – Informações por País.

País	**Usuários**	**Sessões**	**Duração Média da Sessão**
Brasil	707	1.276	00:02:31
Colômbia	93	238	00:01:26
Portugal	5	7	00:00:47
Argentina	4	6	00:02:16
Estados Unidos	4	4	00:00:00
Canadá	2	2	00:00:00
França	2	3	00:00:00
Itália	2	2	00:00:00
Cabo Verde	1	1	00:00:00
Alemanha	1	3	00:02:46
Irlanda	1	1	00:00:00
Sérvia	1	1	00:00:00
Suécia	1	1	00:00:00
Total	**824**	**1.545**	**00:02:19**

Fonte: Elaborada pelos autores (2021).

Informações sobre os tipos de dispositivos mais utilizados também foram computadas através do *Google Analytics*. A Tabela 3 mostra como foi a distribuição por dispositivos. Percebe-se que o número de usuários que acessaram a aplicação pelo *smartphone* foi aproximadamente o dobro dos demais dispositivos. Isso comprova que a responsividade da aplicação foi um fator de sucesso para a experiência e acessibilidade do jogo.

Como mostram as Figuras 11 e 12, no total cerca de 5.650 partidas foram armazenadas no *Firebase* durante o período analisado. A maioria das partidas ocorreu contra o Agente *Sky Connect*, acarretando em um total de 3.520 partidas. O restante das partidas (2.130) foram disputadas contra o agente *Alpha Star*. Em todos os níveis os agentes se sobressaíram de maneira absoluta contra os adversários. No mais, pode-se perceber que para os dois agentes o maior número de partidas foi no nível *Fácil*. É importante lembrar que o nível *Fácil* do agente *Sky*

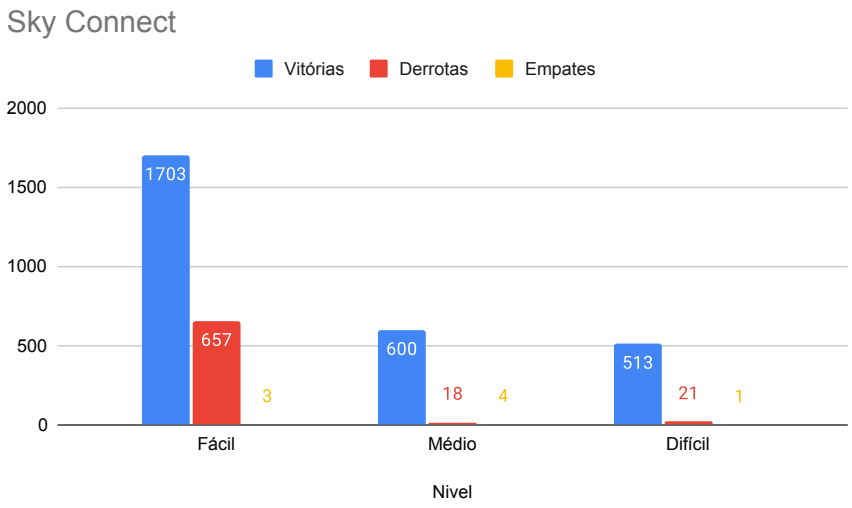
Tabela 3 – Informações por tipo de dispositivo.

Tipo de Dispositivo	Usuários	Sessões	Duração Média da Sessão
Smartphone	541	1.114	00:01:53
Desktop	281	429	00:03:23
Tablet	2	2	00:11:54
Total	**824**	**1.545**	**00:02:19**

Fonte: Elaborada pelos autores (2021).

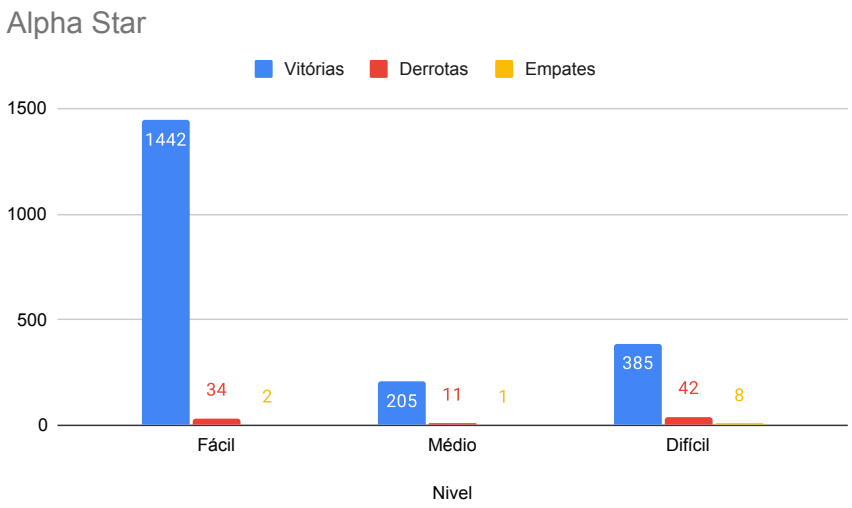
Connect corresponde à profundidade 3 do *minimax* e do agente *Alpha Star* à profundidade 5. Por isso, como esperado, o agente *Alpha Star* possui um número menor de derrotas (34) do que o *Sky Connect*.

Figura 11 – Resultados das partidas para o *Sky Connect*.



Fonte: Elaborada pelos autores (2021).

Figura 12 – Resultados das partidas para o *Alpha Star*



Fonte: Elaborada pelos autores (2021).

6 CONCLUSÕES

Este trabalho apresentou a metodologia para o desenvolvimento de um agente inteligente para o jogo *Connect-4*. A pesquisa englobou o estudo e a aplicação de algoritmos de busca clássicos para encontrar bons movimentos em tempo hábil, sem causar frustrações para os usuários que desafiaram o agente. A aplicação foi disponibilizada na web e atingiu centenas de usuários, resultando em milhares de partidas.

Espera-se que esse trabalho sirva como base para o desenvolvimento de outros agentes para jogos de tabuleiro, facilitando a pesquisa e disponibilização de agentes inteligentes a partir de plataformas *web*.

Como trabalhos futuros, pretende-se adicionar melhorias ao agente através de aperfeiçoamentos no algoritmo *minimax*. O primeiro aperfeiçoamento será o uso de Tabela de Transposição, que consiste em economizar tempo armazenando em *cache* o resultado de cálculos anteriores para evitar o recálculo da pontuação de um estado que já foi explorado durante a busca. A segunda otimização será aprofundamento iterativo, que consiste em explorar a árvore de pesquisa em uma profundidade rasa primeiro e, em seguida, em uma profundidade iterativa, permitindo encontrar o caminho de vitória raso mais cedo e também permitindo manter na tabela de transposição os primeiros resultados das explorações anteriores.

REFERÊNCIAS

ALLIS, V. **A knowledge-based approach of Connect-Four-the game is solved: White wins**. Dissertação (Mestrado), 1988.

ANALYTICS, G. **Google Analytics lets you measure your advertising ROI as well as track your Flash, video, and social networking sites and applications.** 2021. Disponível em: https://analytics.google.com. Acesso em: 19 de Junho 2021.

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

FIREBASE, G. **O Firebase Ajuda as Equipes de Aplicativos para Dispositivos Móveis e da Web a Alcançar o Sucesso**. 2021. Disponível em: https://firebase.google.com/?hl=pt. Acesso em: 03 de Junho 2021.

GIBNEY, E. Google ai algorithm masters ancient game of go. **Nature News**, v. 529, n. 7587, p. 445, 2016.

KANG, X.; WANG, Y.; HU, Y. Research on different heuristics for minimax algorithm, insight from connect-4 game. **J. Intell. Learn. Syst. Appl**, v. 11, n. 02, p. 15–31, 2019.

KORF, R. E. **Artificial intelligence search algorithms**. [S.l.]: Citeseer, 1999.

MISSURA, O.; GAERTNER, T. Online adaptive agent for connect four. In: **Proceedings of the 4th international conference on games research and development cybergames**. [S.l.: s.n.], 2008. p. 1–8.

NODE.JS. **Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.** 2021. Disponível em: https://nodejs.org/en/. Acesso em: 19 de Junho 2021.

NORVIG, P.; RUSSELL, S. **Inteligência Artificial: Tradução da 3a Edição**. [S.l.]: Elsevier Brasil, 2014. v. 1.

REACT, F. **Uma biblioteca JavaScript para criar interfaces de usuário**. 2021. Disponível em: https://pt-br.reactjs.org/. Acesso em: 19 de Junho 2021.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. **Nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

YANNAKAKIS, G. N.; TOGELIUS, J. **Artificial Intelligence and Games**. [S.l.]: Springer, 2018. <http://gameaibook.org>.