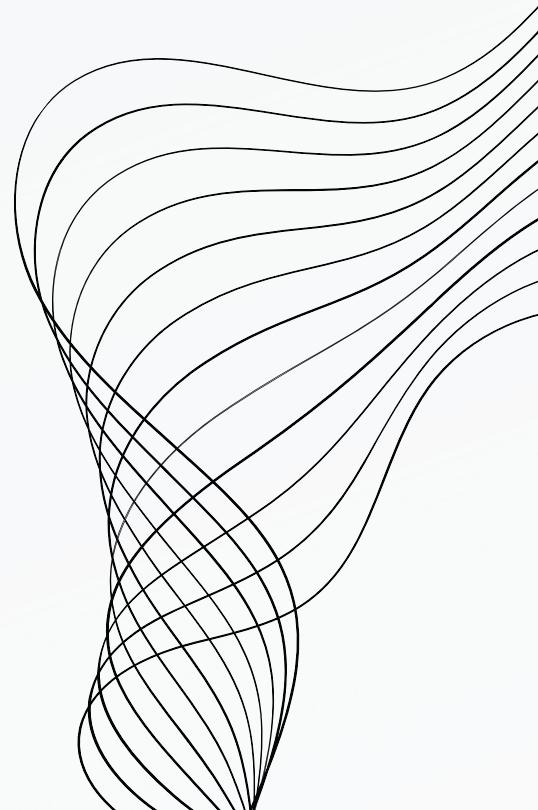


FOCUS

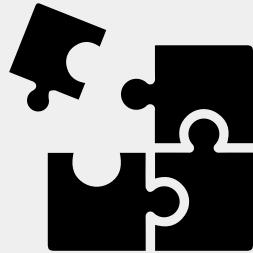
TÓPICO 2C



**PROJETO DE INTELIGÊNCIA ARTIFICIAL
GROUP_A1_7**

ISABEL MOUTINHO	202108767
JOÃO CARDOSO	202108732
LUIZ QUEIROZ	202102362

DEFINIÇÃO DO JOGO



Focus é um jogo de estratégia abstrato em tabuleiro. O jogo é para 2-4 jogadores e seu tabuleiro é em formato de 6x6 com extensões de 1x4 em cada lado. Para o projeto, será admitido 2-4 jogadores com um tabuleiro de tamanho variável.



Os jogadores podem mover pilhas de 1 até 5 peças, somente se a peça mais acima da pilha for da sua cor. As pilhas podem se mover por tantos espaços quanto o número de peças que foram retiradas. É possível unir as pilhas, no entanto, se a união resultar em mais de 5 peças, peças são retiradas debaixo até restarem 5.

Caso as peças retiradas sejam do jogador, elas são guardadas e podem ser posteriormente colocadas no tabuleiro, como uma jogada de reserva, em vez de mover uma pilha. Caso seja do adversário, elas são eliminadas do jogo.



O objetivo do jogo é deixar o adversário sem possibilidade de movimento, ou seja, vence o último jogador que conseguir mover alguma pilha.



TRABALHO RELACIONADO

OPTIMIZAÇÕES MINIMAX

01

ALPHA-BETA PRUNING

02

PRE-SORT MOVES

03

BITBOARDS

04

TRANSPOSITION TABLES

05

BOARD SYMMETRIES

06

REDUZIR JOGADAS POSSÍVEIS

07

MERIFICAÇÃO DE VITÓRIA



Pesquisa das regras do jogo e como o representar usando o pygame.

<https://boardgamegeek.com/filepage/99914/focus-english-rulebook>

<https://www.pygame.org/docs/>



Optimização do minimax devido ao número elevado de caminhos alternativos que o bot pode tomar.

<https://larswaechter.dev/blog/minimax-performance-improvements/>



Pelo facto de haver um elevado número de caminhos que o bot pode percorrer, também pesquisamos outros algoritmos sem ser minimax que lida bem com este problema.

<https://www.youtube.com/watch?v=UXW2yZndl7U>

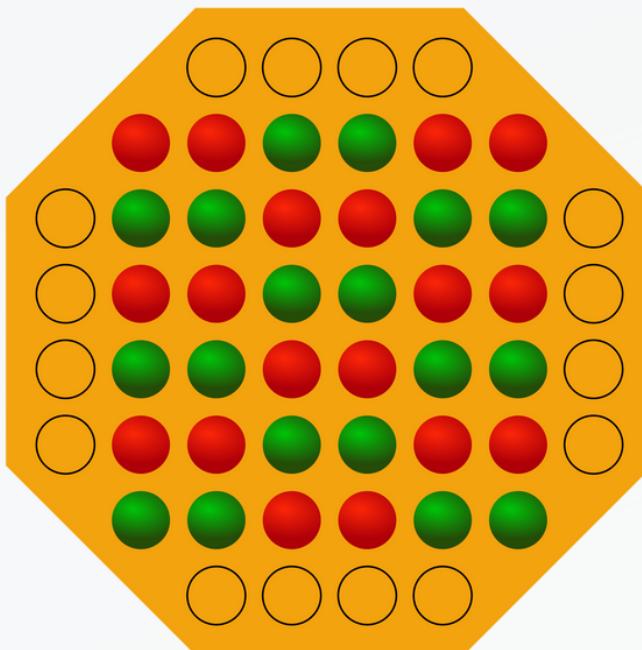
FORMULAÇÃO DO PROBLEMA

O estado do jogo é representado através de um bitmap em que cada espaço tem 17 bits, em que os primeiros dois representam a cor do espaço e os outros 15 as cores das peças nele, sendo que quanto mais para a esquerda maior é a altura das peças na pilha. A cor de cada peça é representada com três bits.

STATE
REPRESENTATION

O jogador inicial é escolhido no menu e o tabuleiro inicia-se como mostrado na imagem

INITIAL STATE



Verifica se o jogador consegue mover uma pilha

OBJECTIVE TEST

FORMULAÇÃO DO PROBLEMA

NAME

Mover pilha para cima, para baixo, para esquerda, para direita e utilizar peça guardada

EFFECTS

- Mover uma pilha em uma direção
- Unir pilhas para formar uma nova pilha
- Retirar peça do tabuleiro, podendo aumentar peças guardadas
- Aumentar número de peças em jogo e reduzir número de peças guardadas

PRECONDITIONS

Para mover uma pilha, a peça do topo precisa de ser da cor do jogador. A quantidade de espaços que pode mexer depende do número de peças que retira da pilha antes de mexer. Só podemos fazer jogadas de reserva se tivermos pelo menos uma peça de reserva e cada pilha só pode ter no máximo 5 peças.

COSTS

- Mover uma pilha para um espaço vazio - 0
- Unir pilhas do mesmo jogador - 1
- Unir pilhas de jogadores diferentes - 3
- Capturar peças adversária - 5
- Capturar peças suas - 10
- Fazer uma jogada de reserva num espaço branco - -5
- Fazer jogada de reserva em sua pilha - 4
- Fazer jogada em pilha adversária - 8

- Comparar o número de peças dos jogadores.
- Comparar o número de peças do topo das pilhas
- Comparar alturas das pilhas.
- Comparar número de peças de reserva de cada jogador.
- Comparar o número de jogadas válidas dos jogadores
- Analisar pilhas sob ameaça de serem unidas

HEURISTICS/EVALUATION FUNCTION

OPERATORS

TRABALHO IMPLEMENTADO

01

02

03

04

AMBIENTE DE TRABALHO

Decidimos programar o projeto em Python com o uso do pygame api. Para partilhar trabalho entre os membros do grupo, recorremos ao git.

ESTRUTURAS DE DADOS

Para representar o estado do jogo recorremos ao uso de um bitmap, onde representamos com bits as cores dos espaços e peças em cada altura. Usamos queues para mudar a vez de quem joga.

INTERFACE

Dispomos de um menu, e painel que vai servir para transmitir informações extra sobre o jogo. Conseguimos interagir com estas interfaces através do uso de botões

MINIMAX

Temos o minimax a funcionar, mas com apenas profundidade de 5-6. Estamos atualmente a implementar medidas de optimização de forma a aumentá-la.

ABORDAGEM

- Associamos um valor a cada jogada que provém de um estado de jogo. Ordenamos as jogadas por ordem decrescente de valor. Assim as jogadas com maior valor são escolhidas primeiro, e vai ser mais provável serem cortadas com o alpha-beta pruning. Também filtramos certas jogadas, que em nenhum caso originam um bom estado de jogo.
- Desenvolvemos 3 heurísticas para avaliar o estado de jogo. Cada heurística corresponde a cada nível de dificuldade do bot e cada dificuldade adiciona algo de novo às dificuldades anteriores

Fácil	Vitória -> 1 e Derrota -> -1
Médio	Compara as peças de reserva e altura das pilhas sob controlo do jogador com as dos outros jogadores e avalia se o estado de jogo é estável ou não.
Difícil	Compara o número de defensores e atacantes de cada peça.

ALGORITMOS IMPLEMENTADOS

Computador Fácil

Utilizamos o algoritmo Monte Carlo para o computador fácil, visto que a heurística deste modo de dificuldade avalia se o estado de jogo é de vitória ou derrota.

Computador Médio

Utilizamos o MiniMax com alpha-beta pruning, mas com várias otimizações:

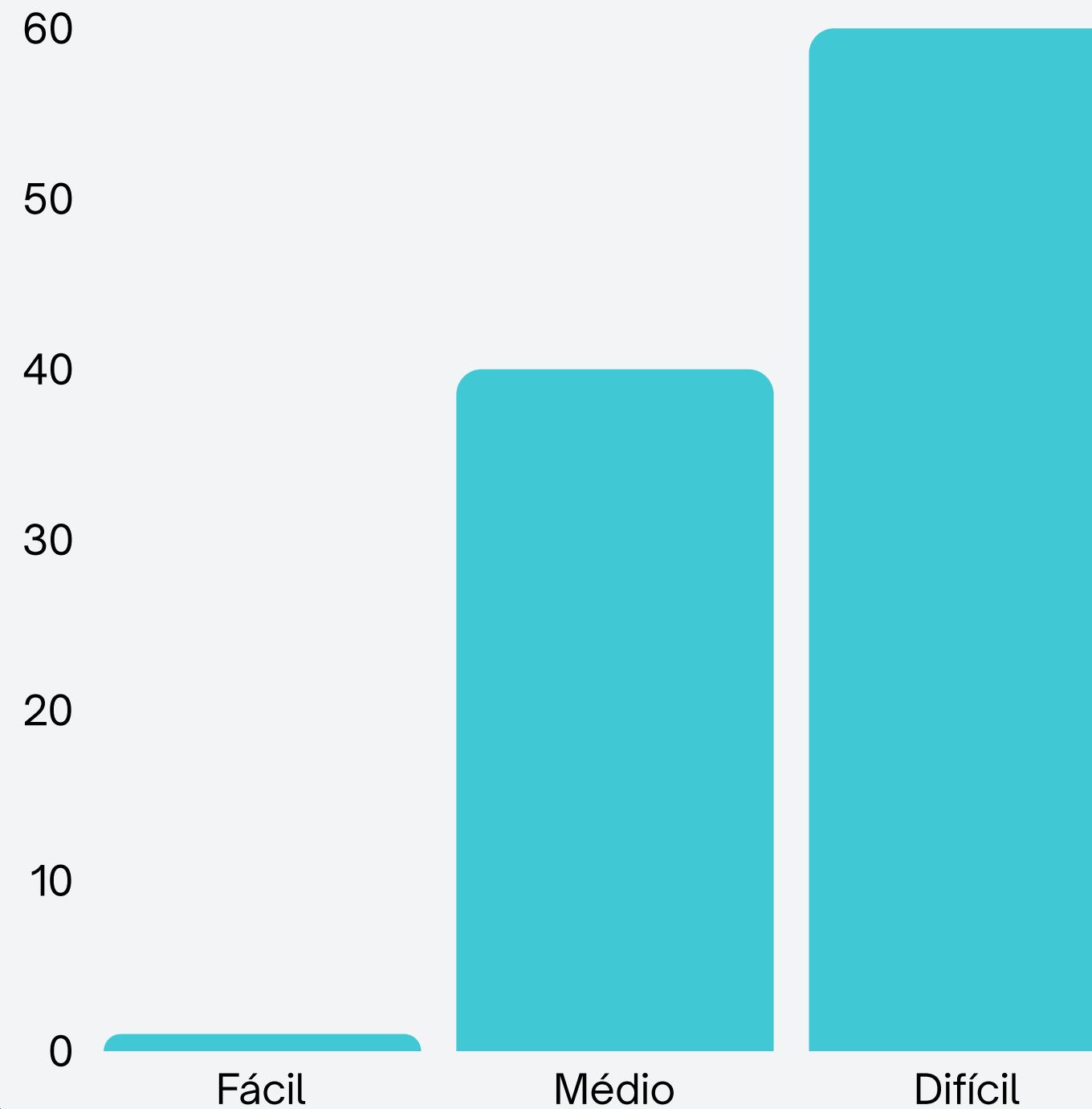
- Ordenação de jogadas que provavelmente originarão um melhor estado de jogo;
- Ver se o estado de jogo é simétrico, se for então só analisar metade das jogadas possíveis;
- Tabela de Transposição que guarda as jogadas prévias, bem como jogadas que originam estados de jogo estáveis;
- Fazer iterative deepening quando uma pessoa deseja ver uma hint;
- Ver se a jogada origina um empate, se sim escolher a segunda melhor jogada se ela originar um estado de jogo favorável.

Computador Difícil

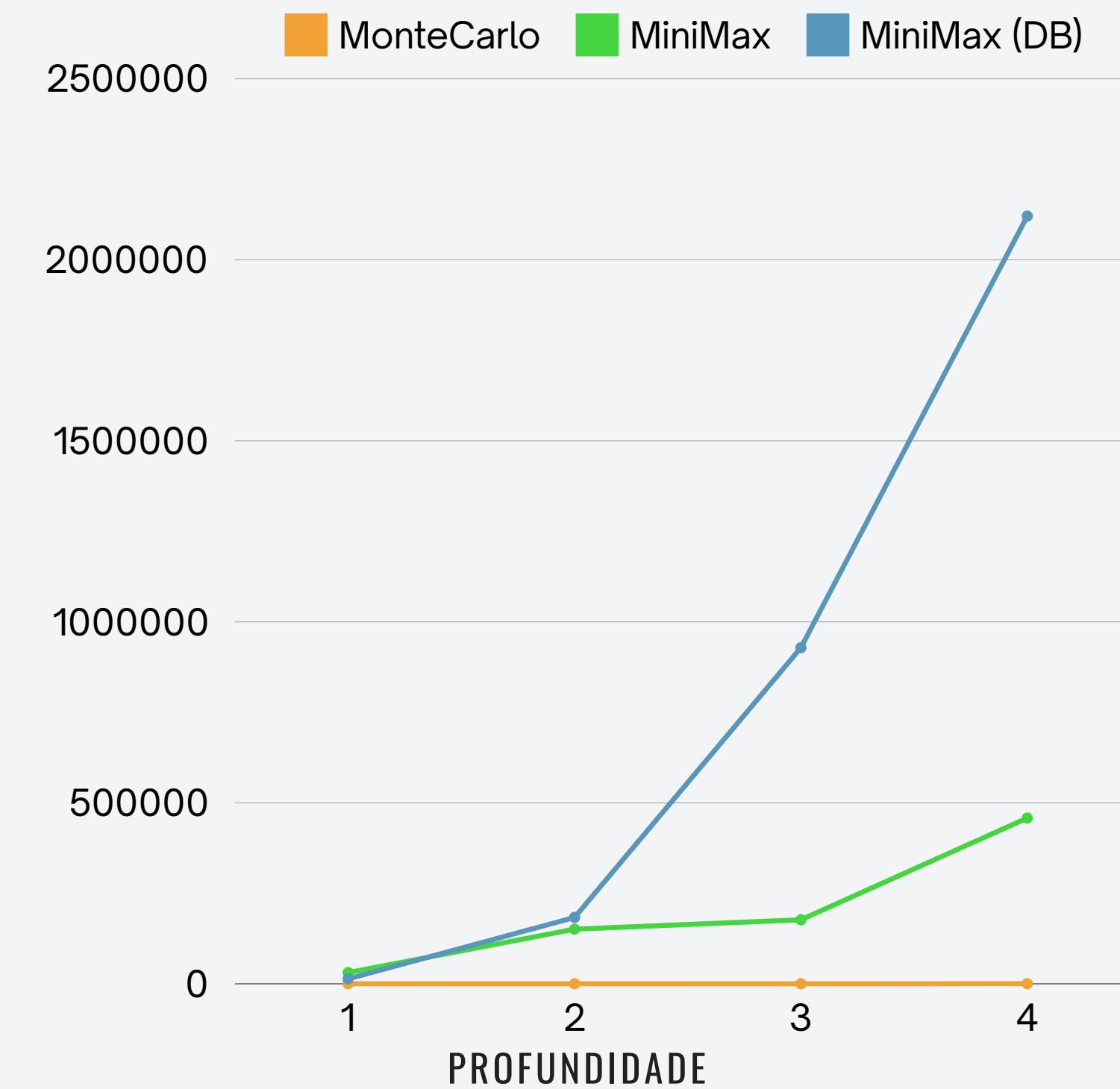
Utilizamos MiniMax sem alpha-beta pruning nem tabela de Transposição, para guardar todas as jogadas de cada estado de jogo numa base de dados, juntamente com o valor dessa jogada e o estado de jogo a que ela origina. Ao longo do tempo quando se adiciona um novo estado de jogo à base de dados, atualiza-se o valor das jogadas que originam esse estado de jogo. Escolhe-se a jogada com maior valor do estado de jogo para ser jogada.

RESULTADOS EXPERIMENTAIS

HEURÍSTICAS DE AVALIAÇÃO
DO ESTADO DE JOGO (EM
MICROSEGUNDOS)



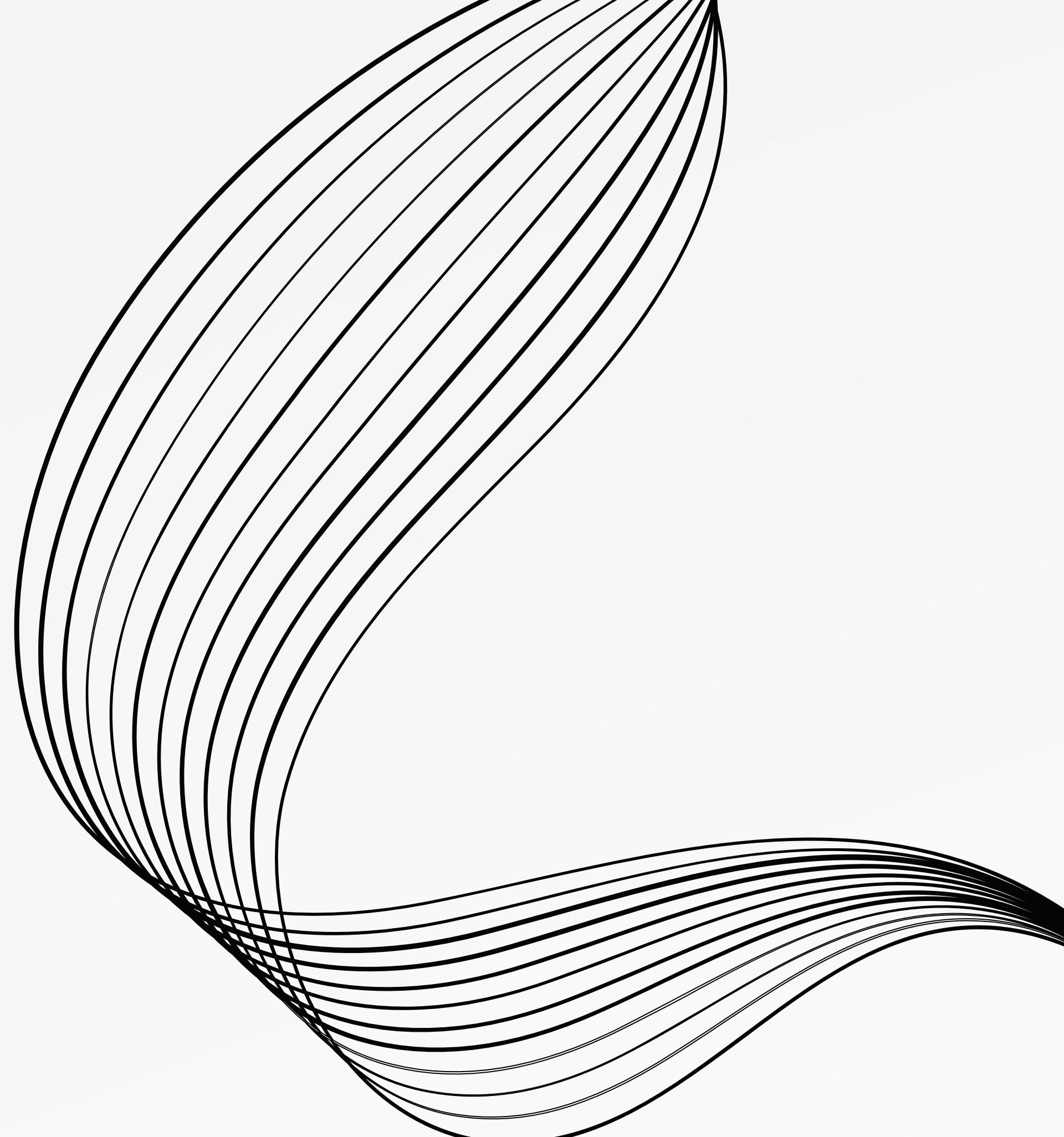
AVALIAÇÃO DO NÚMERO DE ESTADOS DE JOGO
VISITADOS POR SEGUNDO



CONCLUSÃO

Neste projeto tivemos como objetivo implementar o jogo “Focus”, e mais importante ainda conseguir criar um bot que utiliza algoritmos e otimizações de inteligência artificial.

Também sentimos importante tornar o jogo mais flexível ao poder variar o tamanho do tabuleiro e o número de jogadores, bem como a criação de um painel de jogo. Por fim, analisamos as heurísticas e algoritmos feitos de forma a podermos comparar e avaliá-los de forma mais objetiva.



REFERÊNCIAS E MATERIAIS USADOS

01

PYGAME:

<https://www.youtube.com/watch?v=y9VG3Pztok8>

02

OPTIMIZAÇÕES AO MINIMAX:

<https://larswaechter.dev/blog/minimax-performance-improvements/>

03

MONTE CARLO TREE SEARCH:

<https://www.youtube.com/watch?v=UXW2yZndI7U>