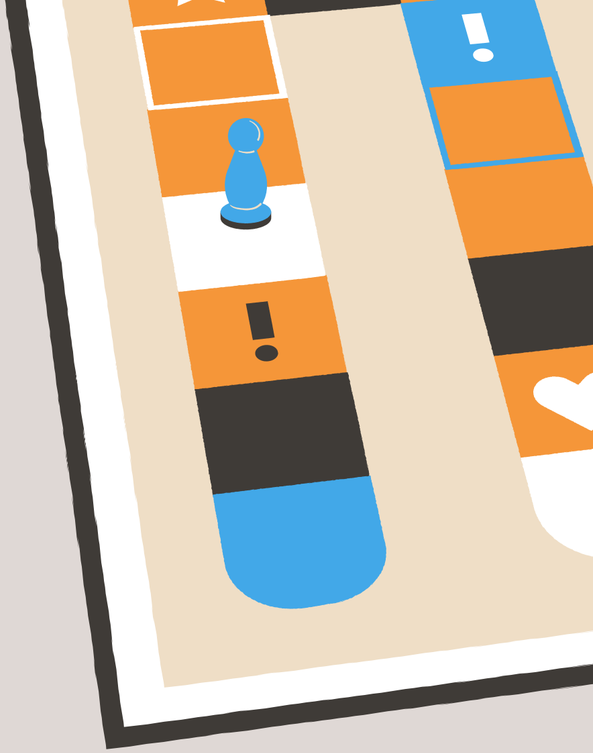
YONMOQUE-HEX



ASSIGNMENT 1

Adversarial Search Methods for Two-Player Board Games

GRUPO A1_63

- Beatriz Pereira, up202207380
- João Silva, up202108713
- Luana Lima, up202206845

ESPECIFICAÇÃO DO TRABALHO

O Yonmoque-Hex é um jogo de tabuleiro para dois jogadores onde o objetivo é **alinhar quatro peças da mesma cor consecutivamente ao mover as peças** já colocadas no tabuleiro, e não apenas ao posicioná-las. O jogador perde se tiver cinco peças consecutivas da sua cor em qualquer momento

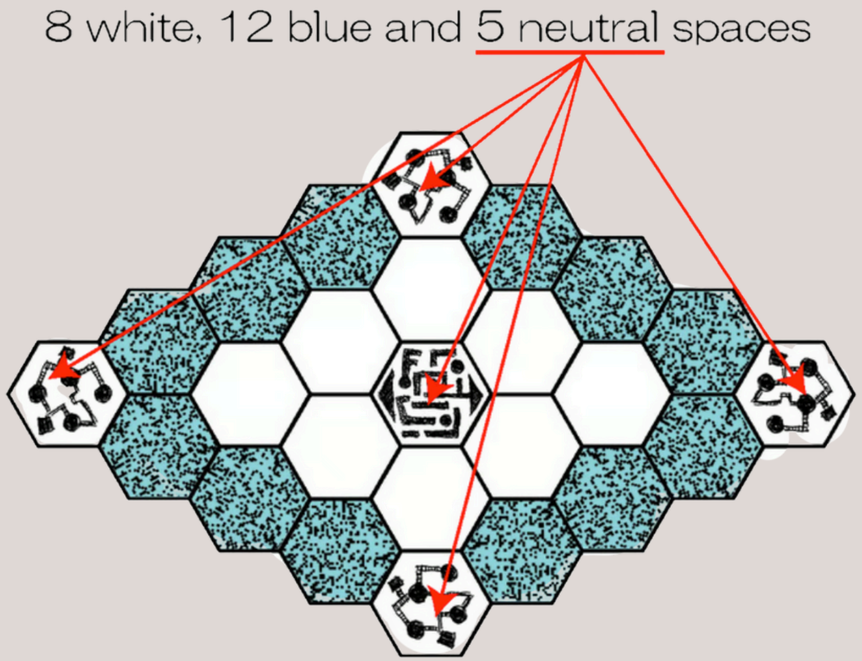


Figura 1. Tabuleiro genérico 5x5

OBJETIVO

Desenvolver uma aplicação que implementa o Yonmoque-Hex, com diferentes níveis de dificuldade para os agentes de IA, utilizando dois métodos principais de pesquisa adversarial: **Humano vs. Humano**, **Humano vs. Computador** e **Computador vs. Computador**

ALGORITMOS A UTILIZAR

- Minimax com αβ pruning;
- Monte Carlo Tree Search (MCTS).

FORMULAÇÃO DO PROBLEMA

REPRESENTAÇÃO DO ESTADO

- **Tabuleiro:** Grafo com nós neutros, brancos e azuis.
- **Células:** Vazia, Azul ou Branca

ESTADO INICIAL

- Tabuleiro vazio
- Definição do jogador inicial (azul ou branco)
- Cada jogador inicia com **6 peças da sua cor**

REGRAS DE MOVIMENTAÇÃO

- Na primeira jogada, o primeiro jogador não pode colocar a sua peça num dos 4 cantos de cor neutra;
- Peças podem **mover-se para qualquer célula adjacente** em qualquer uma das seis direções;
- O movimento que resulta em *"sandwiching"* (encaixar uma peça adversária entre duas peças da própria cor) causa a **inversão da peça adversária**;
- Vencer requer **alinhar quatro peças consecutivas movendo uma peça**, não apenas posicionando-a;
- Um jogador **perde se tiver cinco peças consecutivas na mesma linha** em qualquer momento.

FORMULAÇÃO DO PROBLEMA

OPERADORES

Colocar peça (*insert_piece*)

Pré-condições: Célula vazia, peça disponível
Efeitos: Coloca a peça na célula selecionada

Mover peça *(move_piece)*

Pré-condições: Célula inicial ocupada pela peça do jogador, célula final vazia
Efeitos: Move a peça para uma célula adjacente

Trocar cor da peça *(flip_piece)*

Pré-condições: Peça do adversário está entre duas peças do jogador. Esta posição foi causada ao mover uma peça e não ao colocar uma peça.
Efeitos: A peça do adversário é invertida para a cor do jogador

HEURÍSTICA

- Número de peças conectadas ao caminho mais curto
- Distância mínima até ao objetivo
- Localização no tabuleiro
- Minimização do valor do oponente

CUSTO DE CADA JOGADA

1

IMPLEMENTAÇÃO

A linguagem escolhida para o desenvolvimento do jogo Yonmoque-Hex foi **Python,** sendo utilizada a biblioteca ***PyGame.*** Todo o projeto foi desenvolvido no Visual Studio Code.

O jogo irá tem 3 modos: Humano vs Humano, Humano vs PC e PC vs PC.

Nos modos Humano vs PC e PC vs PC, o utilizador pode escolher entre MiniMax e Monte Carlo, tendo estes 3 dificuldades cada:

- ***Fácil***
- ***Médio***
- ***Difícil***.

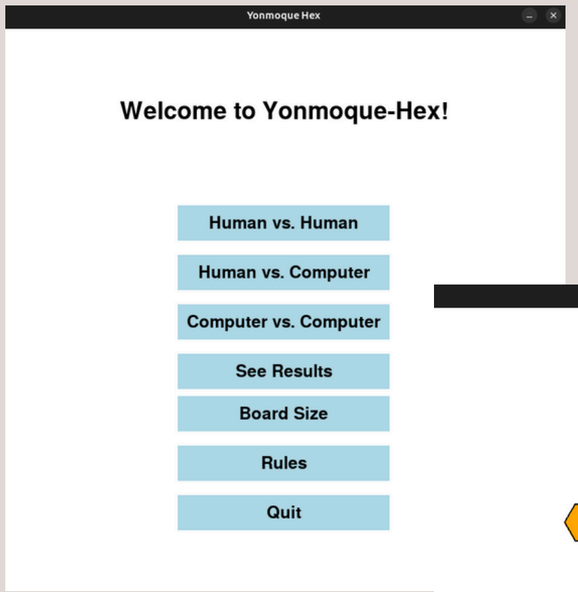


Figura 2. Menu do jogo

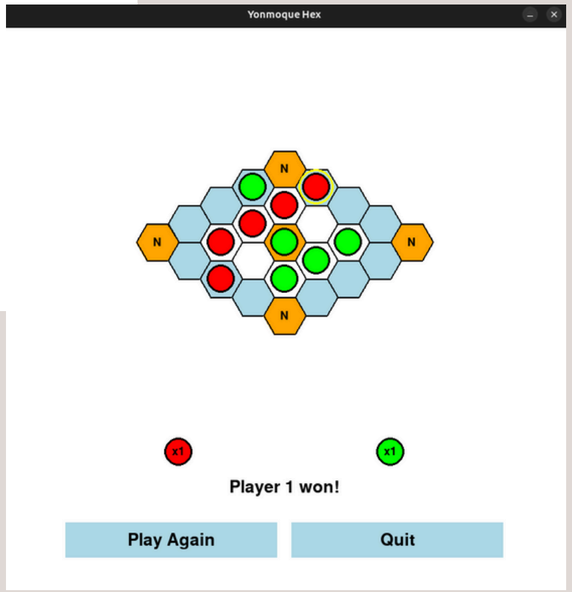


Figura 3. Tabuleiro do jogo

ALGORITMOS
MINIMAX COM ALPHA-BETA PRUNING

MINIMAX

Algoritmo recursivo que explora os estados possíveis até uma determinada profundidade. O jogador atual tenta maximizar a sua pontuação, assumindo que o adversário joga da melhor forma possível (minimizador).

ALPHA-BETA PRUNING

Otimização do Minimax que evita explorar ramos do jogo que não influenciam a decisão final.
Utiliza dois limites:

- **α (alpha)** – melhor valor possível para o **jogador maximizador (IA)**
- **β (beta)** – melhor valor possível para o **jogador minimizador (adversário)**

SELEÇÃO DA MELHOR JOGADA

Caso existam jogadas imediatamente vencedoras, são escolhidas diretamente. Caso contrário, todas as jogadas são avaliadas com Minimax, e é escolhida uma das melhores de forma aleatória.

COMPLEXIDADE

Temporal: O(b^d) **com Alpha-Beta Pruning** → O(b^(d/2))
Espacial: O(b*d)
(sendo b o número médio de jogadas possíveis e d a profundidade do algoritmo)

ALGORITMOS
MONTE CARLO TREE SEARCH (MCTS)

ESTRUTURA DO ALGORITMO

A implementação baseia-se na criação de uma árvore de decisões, onde cada nó representa um estado do jogo.
Cada nó armazena os movimentos possíveis como filhos não testados (*untried moves*).

OPERADORES

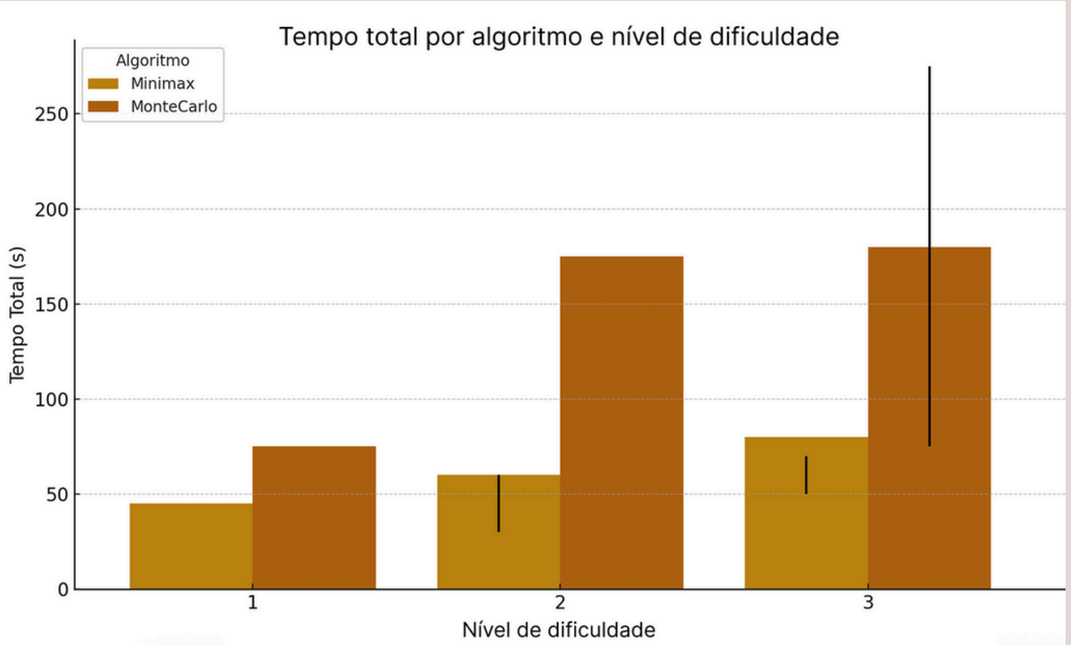
- **Seleção:** Através da fórmula **UCT(Upper Confidence Bound)**, o nó filho com maior valor de UCT é selecionado, equilibrando a exploração de movimentos menos explorados e a exploração de movimentos com melhor desempenho.
- **Expansão: Adiciona novos nós** quando movimentos não foram testados.
- **Simulação:** A partir do novo estado, é realizada uma **simulação aleatória até ao fim do jogo** e no fim é avaliada utilizando uma heurística de avaliação para determinar o vencedor.
- **Backpropagação: Atualiza estatísticas de vitória** nos nós após a simulação.

COMPLEXIDADE

Temporal: O(b^d)
Espacial: O(b*d)
(onde b é o número médio de filhos por nó e d é a profundidade da árvore de decisão)

ANÁLISE DE RESULTADOS

ANÁLISE DO TEMPO TOTAL POR ALGORITMO



Impacto do nível de dificuldade:

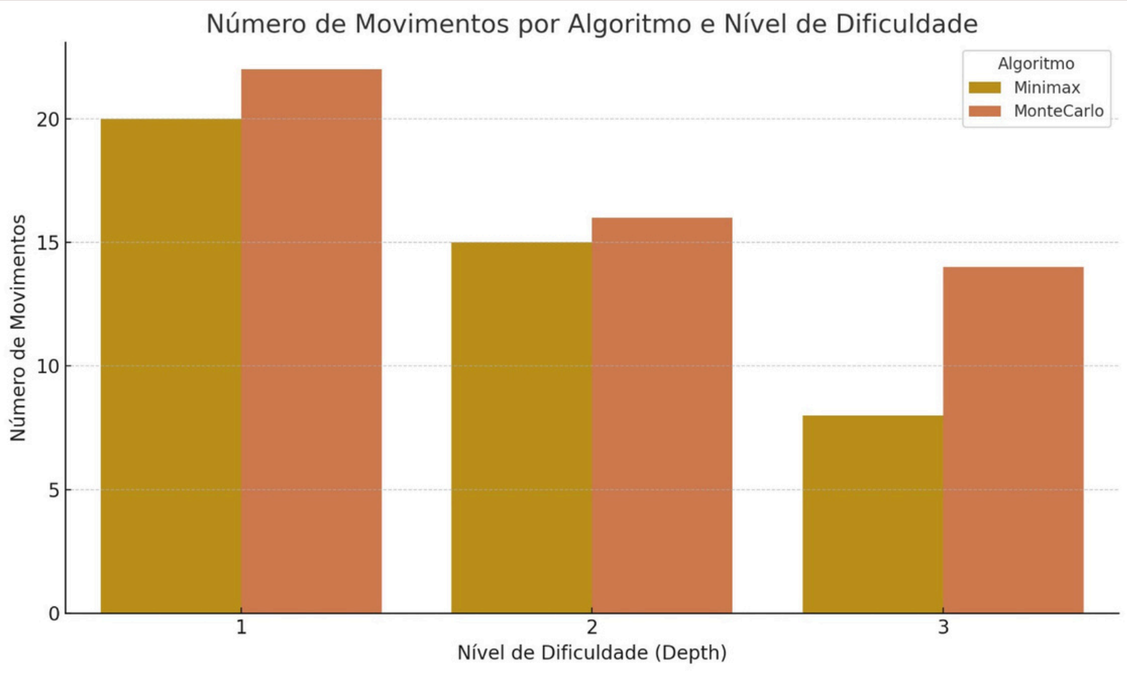
- Minimax mostra um **aumento moderado** e previsível do tempo **com a profundidade** (ex.: profundidade *Easy* vs. *Medium* vs. *Hard*).
- MonteCarlo, por outro lado, tem um **aumento exponencial** no tempo entre níveis, especialmente do nível Easy para o Medium.

Comparação direta entre algoritmos:

- Em confrontos diretos, como Minimax(Medium) vs MonteCarlo(Medium), o **MonteCarlo demora muito mais tempo**.
- Isso sugere que MonteCarlo sacrifica eficiência temporal por decisões potencialmente melhores, o que pode não compensar em todas as situações.

ANÁLISE DE RESULTADOS

ANÁLISE DO NÚMERO DE JOGADAS POR ALGORITMO



Tendência geral entre algoritmos:

- **MonteCarlo** tende a ter partidas com **mais jogadas do que Minimax**, especialmente em confrontos de maior dificuldade.
- Isto pode indicar que o **MonteCarlo promove jogos mais longos e estratégicos**, com menos vitórias rápidas.

Evolução com o nível de dificuldade:

- Em ambos os algoritmos, **o número de jogadas tende a diminuir com a dificuldade**, mas a **diminuição é mais evidente no Minimax**.
- Por exemplo, o confronto MonteCarlo(Medium) vs MonteCarlo(Hard) teve o maior número de jogadas (29), sugerindo um duelo prolongado e mais equilibrado.

Comparação entre algoritmos:

- No duelo direto Minimax(Medium) vs MonteCarlo(Medium), o jogo teve apenas 13 jogadas, um número bastante inferior comparado com jogos entre dois MonteCarlo.
- Isto pode significar que o **Minimax tem uma abordagem mais “decisiva”** ou menos exploratória, terminando os jogos mais rapidamente.

CONCLUSÕES

Com este projeto, foi possível aprofundar o entendimento sobre o funcionamento e comportamento de algoritmos de decisão (Minimax e Monte Carlo Tree Search) aplicados a um jogo competitivo entre dois jogadores.

Relativamente ao algoritmo considerado **mais eficiente**, destacamos o **Minimax**, que apresentou um excelente desempenho em termos de tempo de execução e número reduzido de jogadas, mesmo com o aumento da profundidade.

Por outro lado, o **Monte Carlo Tree Search**, embora apresente jogos mais longos e com decisões aparentemente mais ponderadas, revelou-se **significativamente mais lento**, especialmente a níveis de dificuldade mais elevados.

Concluímos que, no contexto do jogo desenvolvido, o **Minimax representa uma melhor relação entre performance e tempo**, sendo a opção mais adequada para uma IA eficiente. No entanto, o Monte Carlo mostra potencial para simular estratégias mais humanas, sendo interessante para futuras abordagens híbridas ou contextuais.

REFERÊNCIAS

- **Yonmoque Hex | Board Game**
- **Kickstarter - Yonmoque Hex by Mitsuo Yamamoto**