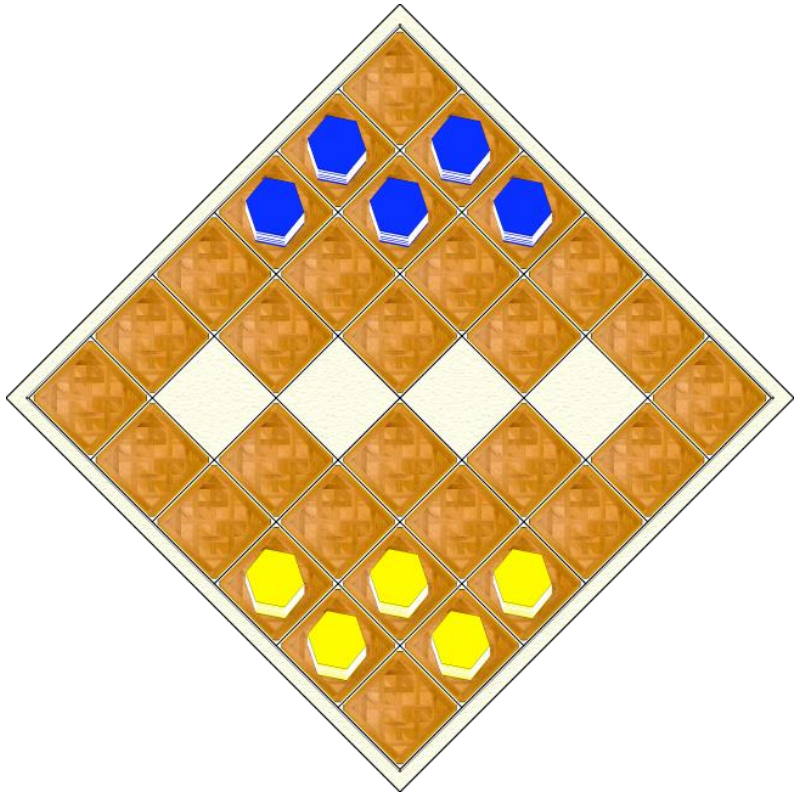
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

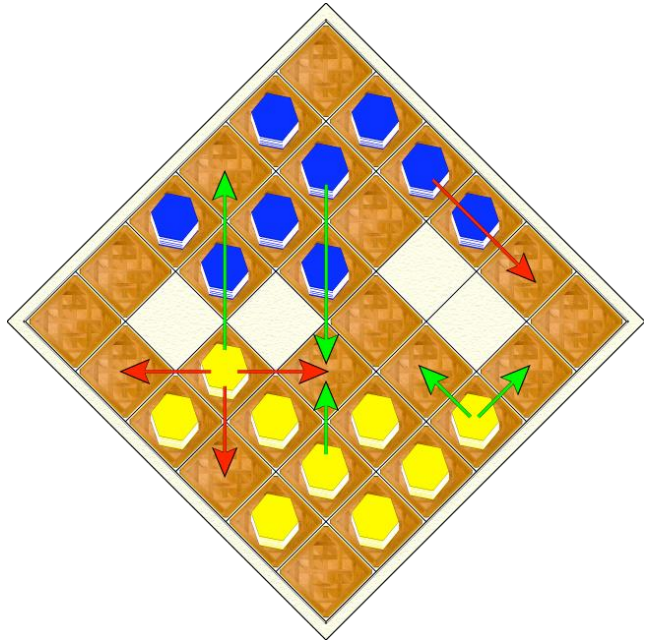
# Jogo de Tabuleiro “Parquet”

Inteligência Artificial 2019/2020



# Objetivo

- Este trabalho tem como objetivo a aplicação do algoritmo de pesquisa MiniMax a um jogo com adversário.
- O jogo escolhido é chamado “Parquet” e trata-se de um simples jogo de tabuleiro cujo objetivo é conseguir colocar três peças no canto do tabuleiro do adversário.
- Cada jogador só pode movimentar as suas peças para a frente e sempre que realiza um movimento pode mexer umas das quatro peças especiais situadas no meio do tabuleiro quando o jogo é iniciado.
- Cada jogada corresponde a dois movimentos.



A verde movimentos válidos e a vermelho movimentos inválidos

# Formulação do Problema como Problema de Pesquisa

- Representação do estado: matriz bidirecional de inteiros de tamanho 6x6 ou 4x4.
- Estado Inicial:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 3 | 3 | 0 |
| 0 | 1 | 0 | 0 | 3 | 3 |
| 0 | 0 | 1 | 0 | 0 | 3 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 2 | 2 | 0 | 0 | 1 | 0 |
| 0 | 2 | 2 | 0 | 0 | 0 |

- Exemplo de um Estado Final:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 2 |
| 0 | 0 | 1 | 3 | 0 | 2 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 3 |

# Formulação do Problema como Problema de Pesquisa

- Operadores A:

| Nome | Pré-Condição | Efeito | Custo | Avaliação de Desempenho |
|---|---|---|---|---|
| Move Right | Espaço à direita livre | Peça desloca-se para a direita | 1 | 1 |
| Move Up | Espaço acima livre | Peça desloca-se para cima | 1 | 1 |
| Move Up Right A | Espaço acima e à direita livre | Peça desloca-se na diagonal para cima | 1 | 2 |
| Move Up Right B | Espaço acima e à direita com uma peça amiga ou inimiga | Peça desloca-se dois espaços na diagonal | 1 | 3 |

Todos os movimentos possíveis para as peças com número “2” no tabuleiro.

# Formulação do Problema como Problema de Pesquisa

- Operadores B:

| Nome | Pré-Condição | Efeito | Custo | Avaliação de Desempenho |
|---|---|---|---|---|
| Move Right | Espaço à direita livre | Peça desloca-se para a direita | 1 | 1 |
| Move Up | Espaço acima livre | Peça desloca-se para cima | 1 | 1 |
| Move Left | Espaço à esquerda livre | Peça desloca-se para a esquerda | 1 | 1 |
| Move Down | Espaço abaixo livre | Peça desloca-se para baixo | 1 | 1 |

Todos os movimentos possíveis para as peças com número “1” no tabuleiro.

# Algoritmos implementados

- Minimax:
  - Função de avaliação: calculo da distância da peça mais próxima à solução.
  - Prioridade a movimentos que fazem a peça cobrir mais distância.
  - Cortes Alfa-Beta.
  - Controlo de profundidade.

```
private double minimax(State state, int[][] board, int depth, int depthLimit, boolean isMax, double alfa, double beta) {
```

  - Devido ao facto do objetivo deste jogo ser apenas chegar ao canto do adversário, não conseguimos pensar em nenhuma outra maneira de avaliar o estado do tabuleiro.
  - Primeiramente implementamos uma versão base do Minimax que só percorria todos os movimentos válidos até à vitório.
  - Posteriormente adicionamos o que foi descrito acima (Prioridade de movimentos, função de avaliação, cortes AlfaBeta, …)
  - Todos estes métodos estão localizados na classe Minimax.

# Resultados Experimentais

- Comparação entre utilização de cortes Alfa-Beta com o mesmo limite de profundidade (neste caso igual a 5):

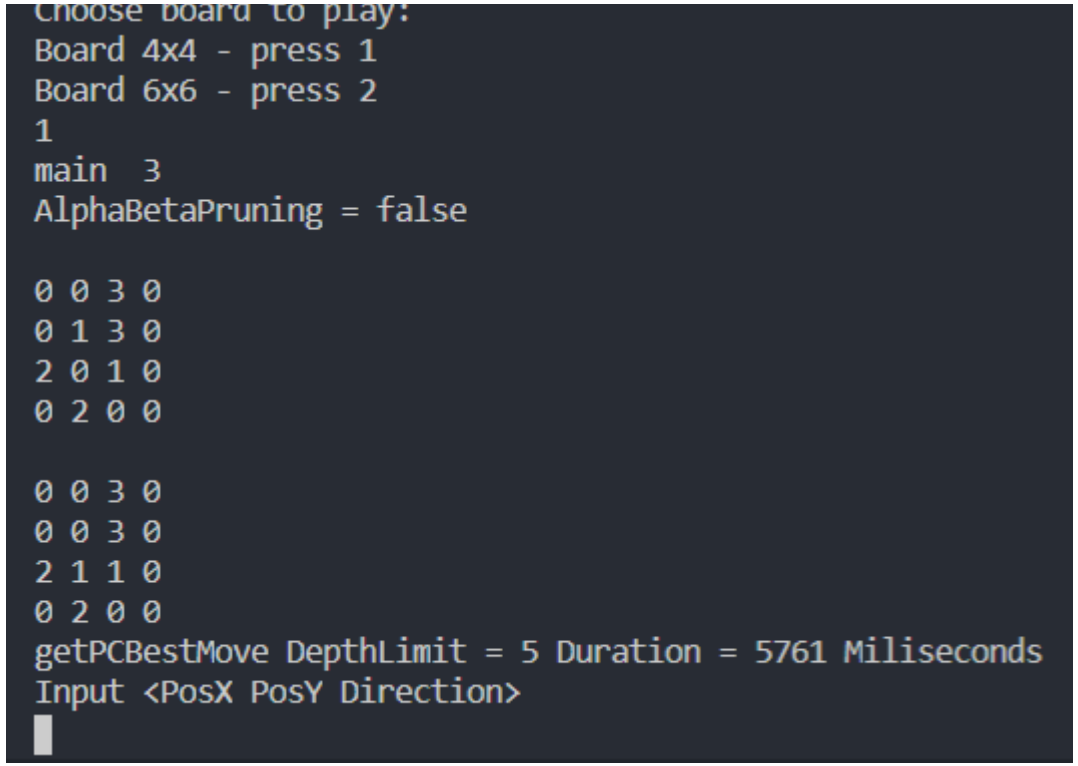
```
Choose board to play:
Board 4x4 - press 1
Board 6x6 - press 2
1
main  3
AlphaBetaPruning = false

0 0 3 0
0 1 3 0
2 0 1 0
0 2 0 0

0 0 3 0
0 0 3 0
2 1 1 0
0 2 0 0
getPCBestMove DepthLimit = 5 Duration = 5737 Miliseconds
Input <PosX PosY Direction>
```

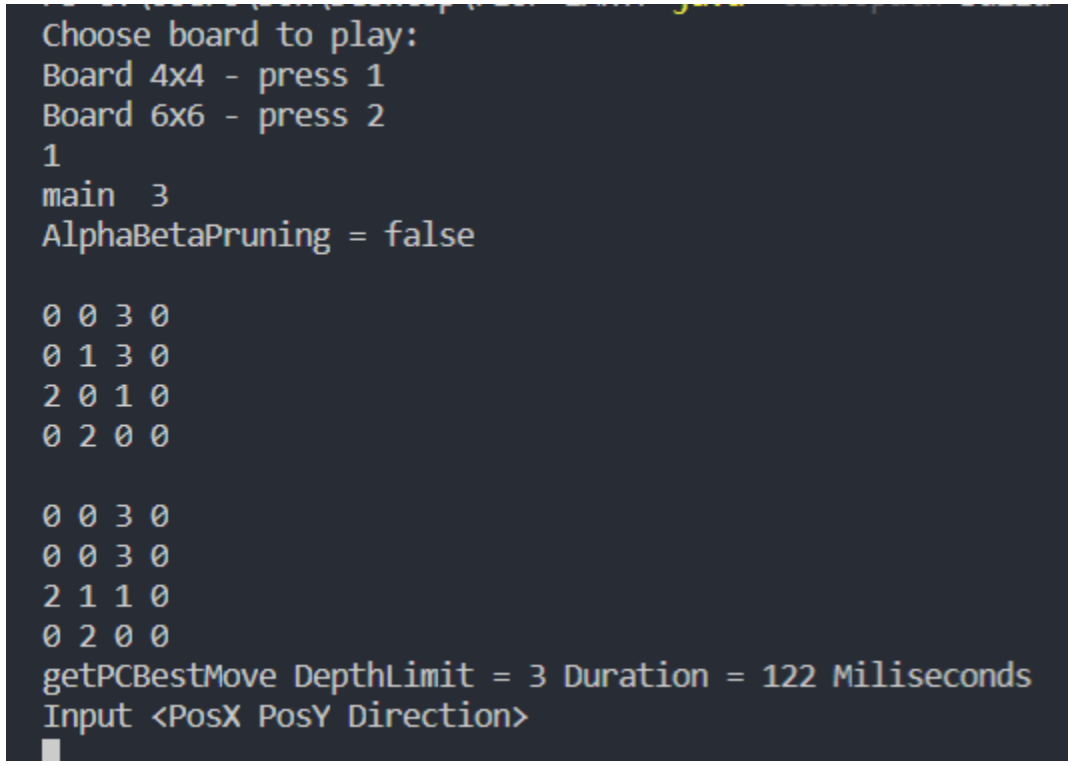
```
Choose board to play:
Board 4x4 - press 1
Board 6x6 - press 2
1
main  3
AlphaBetaPruning = true

0 0 3 0
0 1 3 0
2 0 1 0
0 2 0 0

0 0 3 0
0 0 3 0
2 1 1 0
0 2 0 0
getPCBestMove DepthLimit = 5 Duration = 1355 Miliseconds
Input <PosX PosY Direction>
```

- Como seria de esperar a implementação de cortes Alfa-Beta favoreceu muito a performance do algoritmo.

# Resultados Experimentais

- Comparação entre diferentes níveis de profundidade (neste exemplo 5 comparado com 3):




- Apesar de limitar as decisões do algoritmo, a limitação da profundidade também é essencial para a experiencia do utilizador não ser muito demorada.

# Recursos utilizados

- http://parquet.webstarts.com/
- https://github.com/LazoCoder/Tic-Tac-Toe/blob/master/TicTacToe/ArtificialIntelligence/MiniMax.java
- https://docs.oracle.com/javase/tutorial/uiswing/
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/
- https://www.youtube.com/watch?v=trKjYdBASyQ

# Conclusão

- No final deste projeto conseguimos desenvolver um jogo para dois jogadores, e aplicando o algoritmo Minimax com diferentes variâncias, permitir ao computador realizar jogadas inteligentes que o levam à vitória.
- Implementamos também um modo PC contra PC e Humano contra Humano.