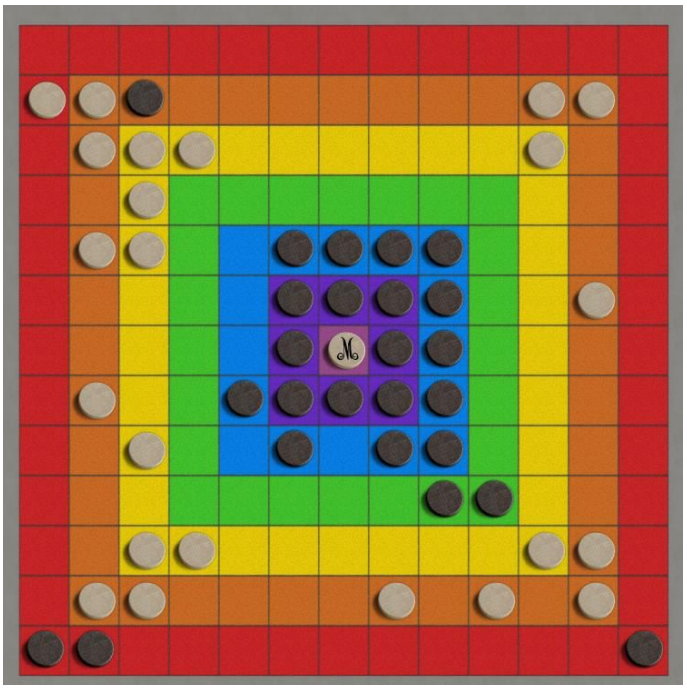
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Pesquisa com Adversários - *Morelli*

Relatório

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação



Elementos do Grupo:

Bernardo José Coelho Leite – 201404464 – up201404464 @gcloud.fe.up.pt
Francisco José Sousa Silva – 201502860 – francisjssilva@gmail.com
Francisco Tomé Macedo Martins Santos Moreira - 201607929 - up201607929@fe.up.pt

20 de maio de 2018

Índice

Objetivo

No âmbito da Unidade Curricular de Inteligência Artificial, o nosso grupo apresenta este relatório com o objetivo de descrever o nosso trabalho.

O **objetivo** do tema escolhido é incidir sobre os tópicos da **Pesquisa Adversarial** com a utilização de **regras de decisão** no contexto da **teoria da decisão**, **teoria do jogo**, **estatística** e **técnicas de minimização de possíveis perdas** para cenários pessimistas (perda máxima). A nossa implementação conta, por isso, com a aplicação do algoritmo ***Minimax*** e derivados para concretizar decisões na presença de incertezas.

Com vista a cumprir os objetivos acima descritos, o grupo escolheu implementar o jogo de Tabuleiro - ***Morelli***. Veja-se agora uma breve descrição das suas regras:

- É possível jogar-se *Morelli* num tabuleiro composto por 13x13 células quadradas que são coloridas em **faixas concêntricas** através das cores do arco-íris;

- Na faixa mais afastada existem 48 quadrados de cor vermelha seguidos pelos da cor laranja, amarela, verde-claro, verde-escuro e roxo. O seguimento destas faixas aponta para a célula central, o **Trono**;

- Cada um dos jogadores começa com 24 peças, dispostas inicialmente na faixa vermelha e em cada uma das jogadas move-se uma peça;

- As peças usufruem da **liberdade das Rainhas no Xadrez** exceto que em cada jogada, estas peças terão de se aproximar para uma faixa mais próxima do trono;

- Uma peça cercada por duas peças adversárias (ortogonalmente ou diagonalmente) é **convertida** numa peça do outro jogador;

- Um jogador que contenha **4 peças dispostas em simetria radial** em relação ao centro, **captura-o**;

- A finalidade do jogo é ter **controlo do centro** quando já não houver mais jogadas possíveis.

Especificação

- **Estados**

Para representar cada estado do jogo utilizamos a estrutura de *int[][] array*, segundo a linguagem de programação *Java*. Para representar as nossas peças em modo texto adotamos o seguinte formato: *0 - posição vazia; 1 - Jogador com Peça de cor Branca; 2 - Jogador com Peça de cor Preta*. Seguem-se os estados de jogo possíveis (estado **inicial**, **intermédio** e **final**):

```
1 1 1 2 1 2 2 1 2 2 1 1 2
2 0 0 0 0 0 0 0 0 0 0 0 1
2 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 0 0 0 2
1 0 0 0 0 0 0 0 0 0 0 0 2
1 0 0 0 0 0 0 0 0 0 0 0 1
2 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 0 0 0 1
2 0 0 0 0 0 0 0 0 0 0 0 2
1 2 2 1 1 1 2 2 2 1 2 1 2
```

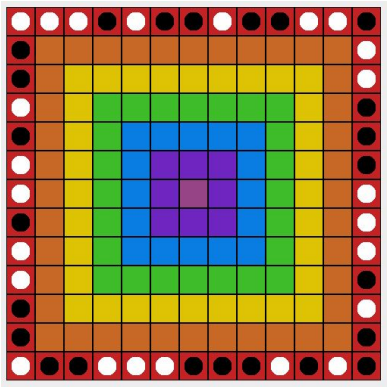


imagem 1 - Estado Inicial com a disposição inicial das peças.

```
0 1 1 2 2 1 0 0 0 0 1 2 0
0 0 0 2 0 0 0 0 0 0 0 0 2
0 0 0 0 0 0 0 2 0 0 0 0 2
1 0 0 0 0 0 0 0 0 0 0 0 2
1 0 0 0 1 0 0 0 1 0 0 0 1
2 0 0 0 0 0 0 0 0 0 0 0 2
1 0 0 0 0 0 1 0 0 0 0 0 1
2 0 2 0 0 0 0 0 0 0 0 0 1
2 0 0 0 1 0 0 0 1 0 0 0 1
1 0 0 0 0 0 0 0 0 2 0 0 2
2 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1
2 1 1 1 2 1 2 2 2 1 2 2 0
```

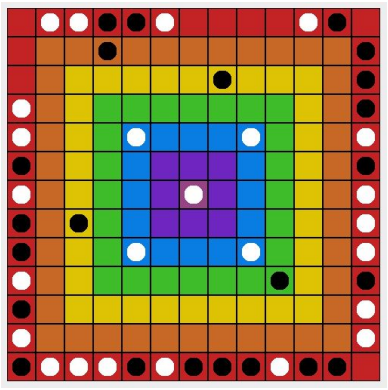


imagem 2 - Estado Intermédio numa situação Captura.

```
0 2 0 0 2 2 0 0 0 0 1 1 0
0 0 0 2 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 2 0 0 0 0 2
2 0 0 0 0 0 0 0 0 2 0 0 1
1 0 0 0 1 0 0 0 2 0 0 0 1
1 0 0 0 0 0 0 2 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 1
2 0 2 0 0 0 0 0 0 0 0 0 2
2 0 0 0 1 0 0 0 1 0 0 0 2
1 0 0 0 0 0 0 0 0 2 0 0 1
2 0 0 0 0 0 0 0 0 0 0 0 2
2 0 0 0 0 0 0 0 0 0 0 0 1
1 2 1 2 2 2 2 1 1 2 1 2 0
```

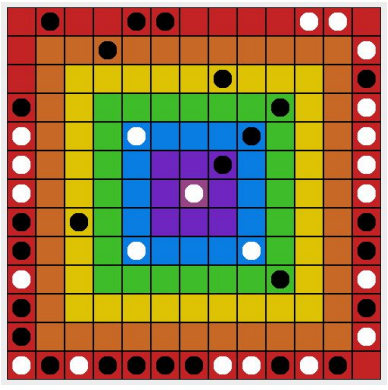


imagem 3 - Estado Intermédio na situação em que uma das Peças é cercada e convertida.

```
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 2 2 0 2 2 2 0 2 0 0 0
0 0 0 2 2 2 2 1 2 0 0 0 0
0 0 0 2 2 2 2 2 1 0 0 0 0
0 0 0 1 1 1 1 2 0 1 0 0 0
0 0 0 1 2 2 1 2 1 0 0 0 0
1 1 0 1 1 1 2 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```

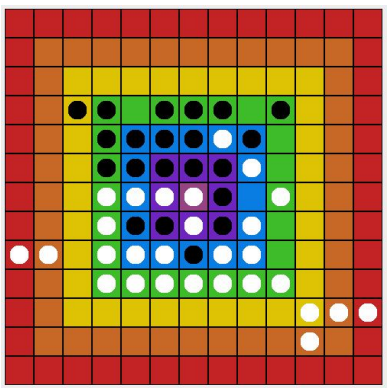


imagem 4 - Estado Final em que o Jogador com as peças brancas é o vencedor (centro capturado e não há mais movimentos possíveis).

- **Função de transição**

Considerar a seguinte máquina de estados finita dada por : $A = (\Sigma, S, So, \delta, F)$

Σ é o Alfabeto de Entrada;
S é o conjunto de estados finitos;
So é o estado inicial, elemento de S;
δ é a função de transição de estados em que ***δ*: *S* x *Σ* → *S***
F é o conjunto de estados Finais.

Tendo em conta que :

Σ = {*capturar*, *bloqueio*, *mover*} e ainda que,

S0 : Estado Inicial com as Peças dispostas na Primeira Faixa (vermelha) do Tabuleiro;
S1 : Estado Intermédio após Movimento de Peça sem que haja qualquer Captura ou bloqueio;
S2 : Estado Intermédio após Captura de uma Peça;
S3 : Estado Intermédio após Bloqueio de uma Peça;
Qualquer um dos estados (S1,S2.S3) **pode ser um estado final** desde que não haja mais movimentos possíveis.

A **função de transição** define-se pelo seu conjunto de transições: ***δ(S0,mover) = S1, δ(S1, capturar) = S2, δ(S1, bloqueio) = S3, δ(S2, mover) = S1 e δ(S3, mover) = S1***.

- **Função de avaliação**

Apresentamos a seguinte função heurística de avaliação que pondera a soma de vários fatores através da sua influência no valor da posição.

heurística = 1 ***(para vitória)*** **e** **heurística = -1** **(para derrota)**

Se Jogador estiver em modo Ofensivo:

heurística = Captura * 0.4 + Bloqueio de Captura * 0.3 + Cercar Peça * 0.20 + Bloquear Cerco de Peça * 0.10

Se Jogador estiver em modo Defensivo:

heurística = Bloqueio de Captura * 0.4 + Captura * 0.3 + Cercar Peça * 0.20 + Bloquear Cerco de Peça * 0.10

O modo **Ofensivo** é quando o jogador atual tem como prioridade Capturar o centro;
O modo **Defensivo** é quando o jogador atual tem como prioridade Bloquear a Captura do Centro;

Cercar Peça = ($\frac{1}{6}$ * Nr de peças convertidas em resultado do Cerco)

1 <= Nr Peças Convertidas em Resultado do Cerco <= 6

Bloquear Cerco de Peça = ($\frac{1}{6}$ * Nr de peças convertidas em resultado do Cerco)

1 <= Nr Peças Convertidas em Resultado do Cerco <= 6

- **Algoritmos de pesquisa a aplicar**

Os algoritmos de pesquisa utilizados são o ***Minimax*** e o ***Minimax*** **com Cortes Alfa-Beta**. Cada um deles utiliza na sua implementação a função heurística anterior para estimar o valor heurístico de um determinado nó.

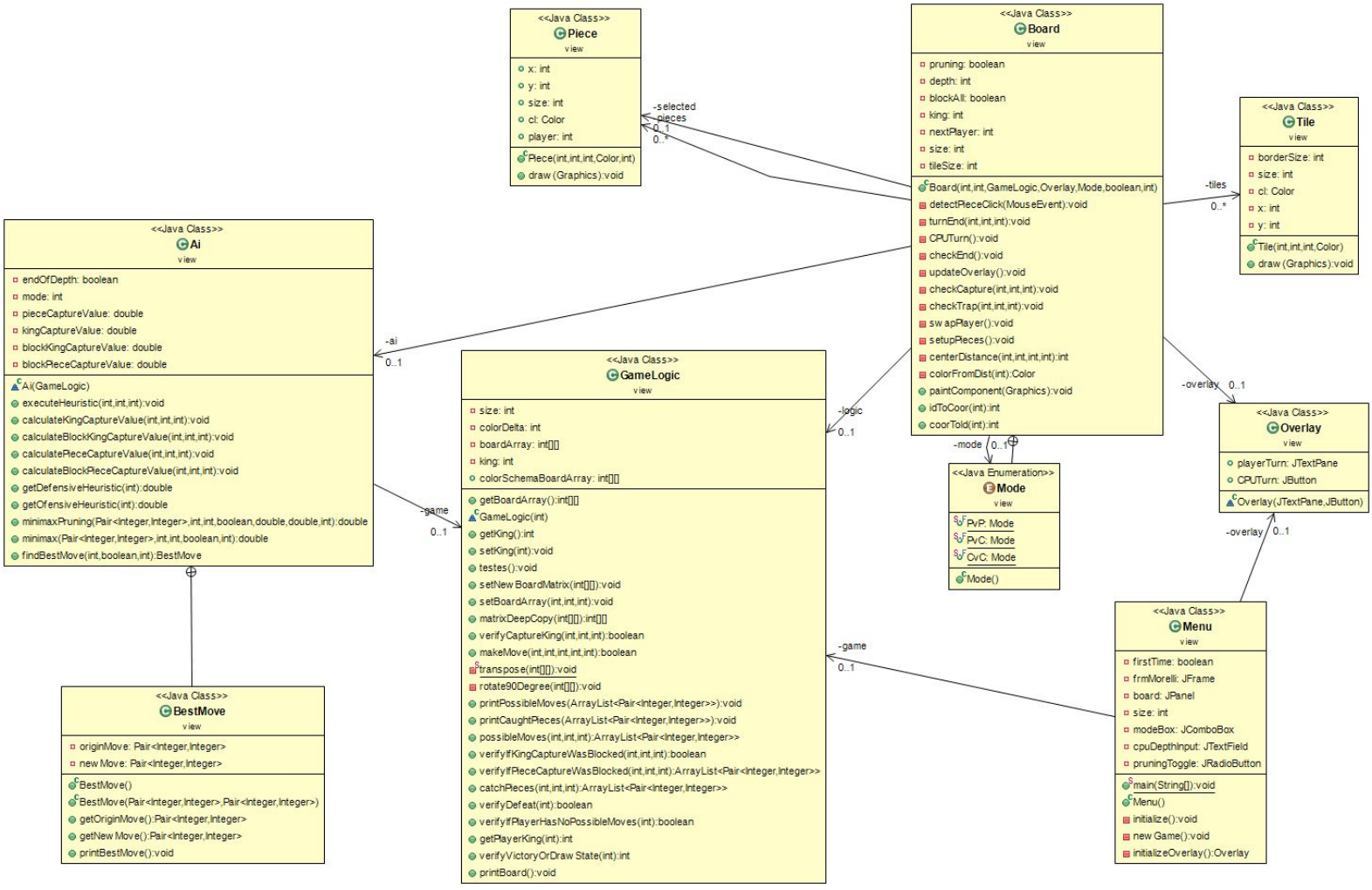
Desenvolvimento

- **Ambientes de Desenvolvimento**

  Eclipse através do *Windows*;
  IntelliJ IDEA através do *UNIX*.

- **Estrutura da Aplicação**

  Diagrama de Classes:



  Descrição dos Módulos e das Classes:

  - Relativamente à implementação da Lógica de jogo:

    GameLogic.java - Classe que contém todos os métodos que fazem cumprir as regras do jogo.

  - Relativamente à implementação da Inteligência Artificial:

    Ai.java - Classe com os métodos responsáveis pela implementação da Inteligência Artificial no que diz respeito aos algoritmos Minimax e derivados e ainda as experiências feitas para comprovar a sua eficácia.

  - Relativamente à implementação da Interface Gráfica:

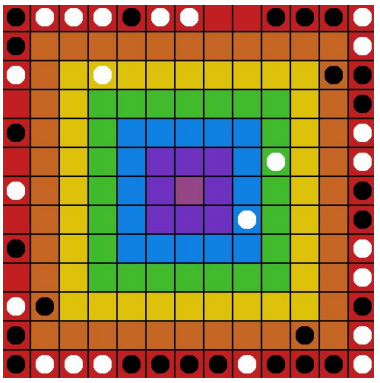
    Menu.java - O Menu da nossa aplicação.
    Board.java - Representação do tabuleiro de jogo.
    Piece.java , Tile.java , Overlay.java - Componentes gráficos do jogo.

Experiências

Notas: *- As seguintes experiências podem ser verificadas no código fonte do Programa.*
- Os tempos indicados para cado caso resultam da média entre 3 medições.

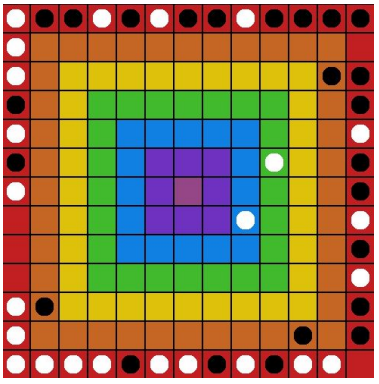
Experiência 1 - Captura do Centro: *Considerar o cenário em que o jogador atual (maximizante) tem a opção de mover uma das suas peças para uma posição do tabuleiro em que se efetua uma Captura do Centro.*
Objetivo: *Retornar como melhor jogada aquela que corresponde à posição em que se efetua a Captura do Centro.*
Resultado:

	Minimax			**Minimax com cortes Alfa-Beta**		
Profundidade	**1**	**2**	**3**	**1**	**2**	**3**
Resultado (melhor jogada):	De (1,0) para (1,2)	De (1,0) para (1,2)	De (1,0) para (1,2)	De (1,0) para (1,2)	De (1,0) para (1,2)	De (1,0) para (1,2)
Tempo (s) :	0.036	1.446	233.61	0.033	1.361	15.944



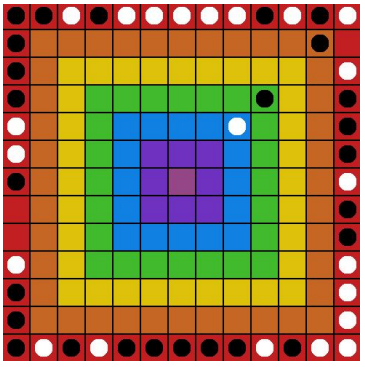
Experiência 2 - Impedir Captura do Centro: *Considerar o cenário em que o jogador atual (max) tem a opção de mover uma das suas peças para uma posição do tabuleiro em que impede que o Centro seja Capturado.*
Objetivo: *Retornar como melhor jogada aquela que corresponde à posição em se impede que o Centro seja Capturado.*
Resultado:

	Minimax			**Minimax com cortes Alfa-Beta**		
Profundidade	**1**	**2**	**3**	**1**	**2**	**3**
Resultado (melhor jogada):	De (0,3) para (1,2)	De (0,3) para (1,2)	De (0,3) para (1,2)	De (0,3) para (1,2)	De (0,3) para (1,2)	De (0,3) para (1,2)
Tempo (s) :	0.032	1.135	242.19	0.0257	1.043	15.244



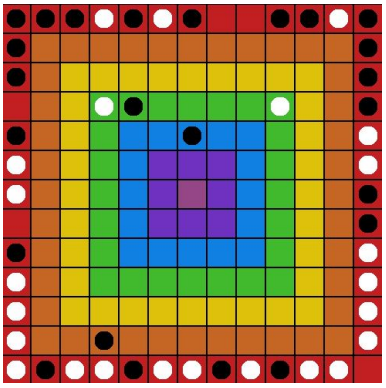
Experiência 3 - Cercar Peça: *Considerar o cenário em que o jogador atual (max) tem a opção de mover uma das suas peças para uma posição do tabuleiro em que irá cercar uma das peças do adversário.*
Objetivo : *Retornar como melhor jogada aquela que corresponde à posição em que é possível cercar uma das peças do adversário.*
Resultado:

	Minimax			**Minimax com cortes Alfa-Beta**		
Profundidade	**1**	**2**	**3**	**1**	**2**	**3**
Resultado (melhor jogada):	De (0,8) para (2,10)	De (0,8) para (2,10)	De (0,8) para (2,10)	De (0,8) para (2,10)	De (0,8) para (2,10)	De (0,8) para (2,10)
Tempo (s) :	0.035	1.326	245.32	0.029	1.200	17.962



Experiência 4 - Decisão dependente da Profundidade: *Considerar o cenário em que o jogador atual (maximizante) tem a possibilidade de jogar para uma posição em que cerca uma peça do oponente. No entanto, depois de a cercar existe a possibilidade do adversário cercar uma das suas peças.*
Objetivo: *Retornar como melhor jogada aquela que corresponde à melhor posição de acordo com a profundidade estabelecida. Se* ***prof < 3 jogará para (3,5)*** *mas* ***se prof >=3 jogará para (9,3) ou (9,9)*** *pois nestas se pode concretizar uma Captura do Centro e simultaneamente evitar que a sua peça seja cercada.*
Resultado:

	Minimax			**Minimax com cortes Alfa-Beta**		
Profundidade	**1**	**2**	**3**	**1**	**2**	**3**
Resultado (melhor jogada):	De (0,5) para (3,5)	De (0,5) para (3,5)	para (9,3) ou (9,9)	De (0,5) para (3,5)	De (0,5) para (3,5)	para (9,3) ou (9,9)
Tempo (s) :	0.0263	1.046	231.42	0.0256	1.015	12.229

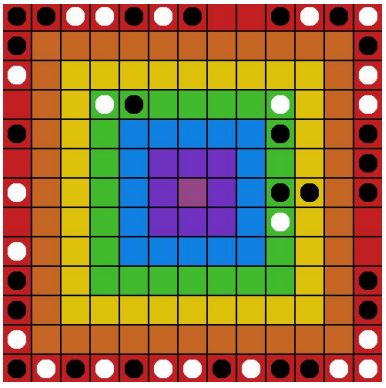


Experiência 5 - Cercar 1 ou 2 peças: *Considerar o cenário em que o jogador atual (maximizante) tem as seguintes opções:*

1. *Colocar uma peça de forma a Cercar uma das peças do Adversário;*
2. *Colocar uma peças de forma a Cercar duas peças do Adversário.*

Objetivo: *Retornar como melhor jogada aquela que corresponde à posição que permite cercar duas peças do adversário.*
Resultado:

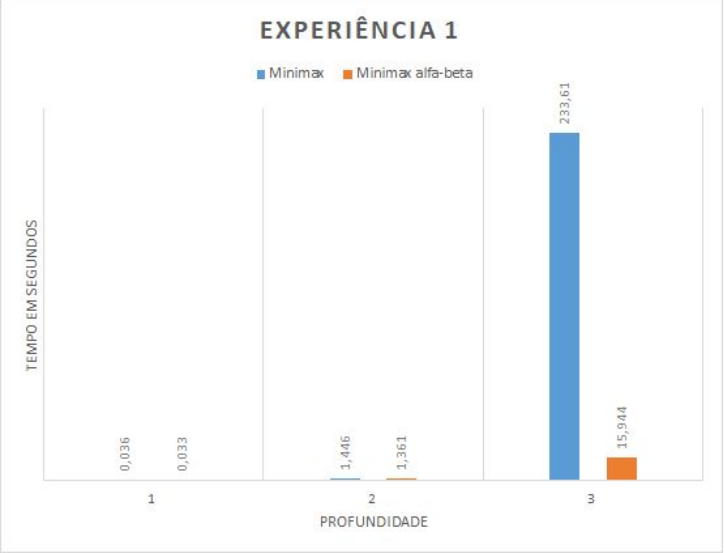
	Minimax			**Minimax com cortes Alfa-Beta**		
Profundidade	**1**	**2**	**3**	**1**	**2**	**3**
Resultado (melhor jogada):	De (2,12) para (5,9)	De (2,12) para (5,9)	Varia	De (2,12) para (5,9)	De (2,12) para (5,9)	Varia
Tempo (s) :	0.0273	0.909	239.13	0.0213	0.887	14.172



Conclusões

Os resultados das experiências anteriores correspondem ao que teoricamente deveria suceder-se ao utilizar-se os algoritmos Minimax e Minimax com Cortes Alfa-Beta, ou seja, para o Minimax os tempos de pesquisa de uma solução são inferiores comparativamente aos tempos de pesquisa utilizado a variação com Cortes Alfa-Beta. Para além disto, o estabelecimento de uma profundidade máxima é também um fator crucial no quão demorado poderá ser a pesquisa.

Observe-se os seguintes gráficos que de uma forma clara mostram a diferenças que podem existir dependendo de vários fatores:



Melhoramentos

Depois de analisar os resultados das experiências feitas para vários cenários do nosso jogo constatamos que é possível otimizar o tempo de pesquisa de uma solução, neste caso, a melhor jogada para um determinado jogador e estado de tabuleiro. Tendo em conta os algoritmos implementados na nossa aplicação (Minimax e Minimax com Cortes Alfa-Beta) consideramos pertinentes para futuro as seguintes melhorias:

- **Reduzir o fator de ramificação**, através da seleção das melhores jogadas possíveis para uma determinada peça;
- **Aplicar uma tabela de transposição**;
- **Aplicar uma estratégia do tipo Aprofundamento Progressivo** , através do aumento progressivo da profundidade até que se alcance um determinado tempo, abandonando assim a estratégia de se estabelecer uma profundidade desde ínicio.

Recursos

Bibliografia

- *Apontamentos das aulas (ppt);*
- *"Artificial Intelligence: A modern approach". S.Russel, P.Norvig, Prentice-Hall, 3rd Edition, 2010;*
- *"Artificial Intelligence for Games". Ian Millington, John Funge, 2nd Edition, 2009;*
- Link*: http://www.boardspace.net/english/about_morelli.html*.

Software e Tecnologias

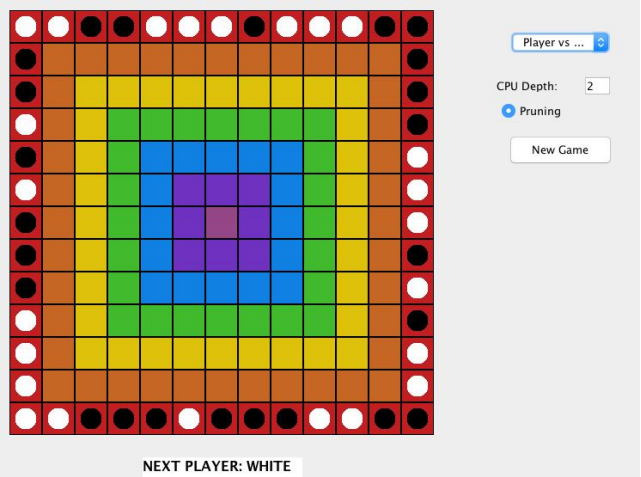
- Utilização da Linguagem de Programação *Java*;
- Utilização da *widget toolkit Swing*;
- Utilização do Ambiente de Desenvolvimento Integrado 1: *Eclipse*;
- Utilização do Ambiente de Desenvolvimento Integrado 2: *IntelliJ IDEA*.

Percentagem aproximada de trabalho efectivo de cada elemento do grupo:

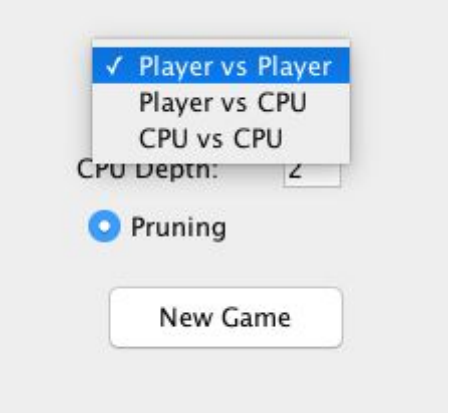
Bernardo José Coelho Leite – 33.3%
Francisco José Sousa Silva – 33.3%
Francisco Tomé Macedo Martins Santos Moreira - 33.3%

Apêndice - Manual do utilizador

O seguinte menu corresponde à interface do nosso jogo.

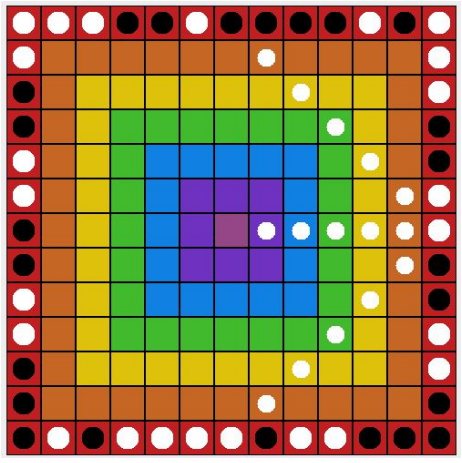


É possível escolher-se o modo de jogo através do seguinte sub-menu:



Para o modo de jogo CPU vs CPU pode-se indicar a profundidade máxima pretendida relativamente ao algoritmo Minimax. Para isso digitar o número pretendido em "CPU Depth". A opção "Pruning" serve para se utilizar o algoritmo Minimax com Cortes Alfa-Beta.

É possível ver-se os movimentos possíveis para uma determinada peça quando esta for selecionada:



Para mover uma peça basta selecionar a posição de origem e depois a posição de destino.