Aboyne

Relatório Intercalar

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

José Ricardo de Sousa Coutinho – ei12161 – ei12161@fe.up.pt

Leonel Jorge Nogueira Peixoto – ei12178 – ei12178@fe.up.pt

19 de Abril de 2015

Índice

Objectivo

Este projeto consiste na implementação de um jogo de tabuleiro com o algoritmo depesquisa *Mini-Max* com *Cortes Alfa-Beta* e heurísticas apropriadas de forma a gerar oponente(s) inteligente(s).

A aplicação de conceitos de inteligência artificial nos algoritmos utilizados possibilitará um aumento do nível de dificuldade para um adversário inteligente.

Com a realização deste trabalho pretende-se por em prática as competências teóricas adquiridas, na unidade curricular de Inteligência Artificial.

Especificação

O Jogo

O *Aboyne* é um jogo de 2 jogadores num tabuleiro hexagonal de 5 por 5 células.

No estado inicial do jogo cada jogador tem 9 pedras e uma célula objetivo, como apresentado na **Figura 1**.

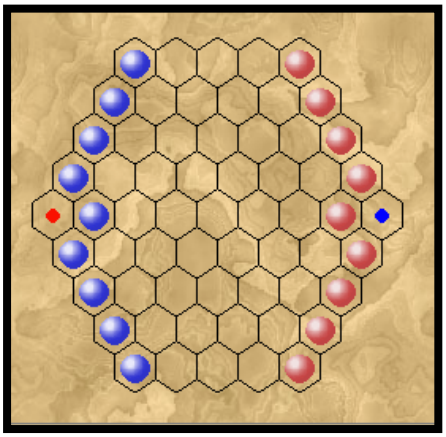


Figura 1 - Estado inicial do jogo.

Regras

O jogo é jogado por turnos, em que cada jogador faz apenas uma movimentação de uma das suas pedras não bloqueadas.

Uma pedra fica bloqueada se estiver adjacente a uma pedra do adversário.

Uma pedra pode ser movida para uma célula vazia adjacente ou saltar por cima de uma linha de pedras amigas ocupando a célula imediatamente a seguir. Se essa célula estiver ocupada por uma pedra adversária, então essa pedra será capturada.

Uma pedra não se pode mover para uma célula objetivo do adversário.

As pedras de cada jogador começam sempre em lados opostos pré-determinados.

A célula objetivo de um jogador encontra-se sempre no lado oposto desse jogador.

Na **Figura 2** as pedras marcadas estão bloqueadas. Se for a vez do jogador vermelho, esse pode capturar a pedra azul bloqueada movendo a sua pedra não bloqueada por cima da sua bloqueada.

Se for a vez do jogador azul, este pode capturar a pedra vermelha bloqueada com uma das duas pedras não bloqueadas que estão alinhadas com as vermelhas. Depois dessa captura o jogador vermelho não terá mais jogadas possíveis pois a sua única pedra acabou de ser bloqueada.

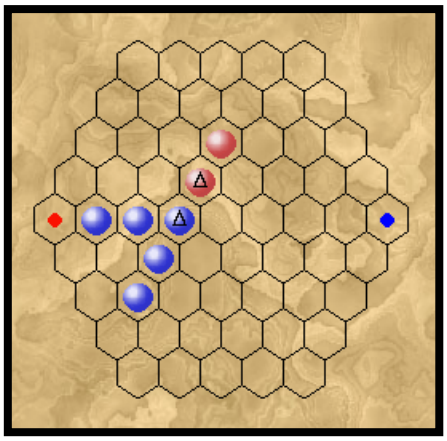


Figura 2 – Exemplo de Jogadas.

Condições de vitória

O jogo acaba e dá a vitória ao jogador que coloca uma das suas pedras na célula objectivo respectiva ou quando o seu oponente não tem mais movimentos possíveis.

Na **Figura 3**, se o jogador azul mover a pedra para a célula 1, o jogador vermelho irá mover a sua pedra não bloqueada para a célula 2 bloqueado assim a pedra azul na célula 1. Como consequência o jogador azul não tem mais jogadas possíveis e o jogador vermelho ganha.

O jogador azul deverá jogar para a célula 3 e de seguida para a 4 de forma a capturar a pedra vermelha marcada. Como tal o jogador azul estará numa posição vencedora porque o jogador vermelho não tem mobilidade suficiente para contra-atacar.

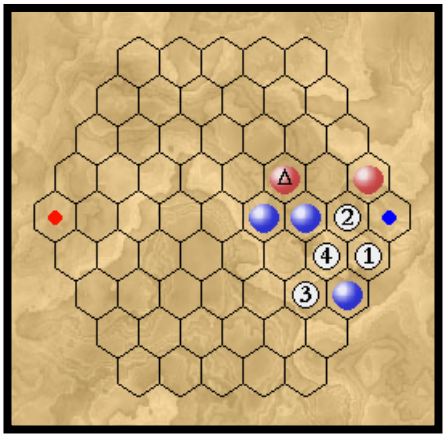


Figura 3 – Jogadas finais.

Complexidade

Para um jogo típico, num tabuleiro 5x5, cada jogador começa com 9 pedras cada e na primeira jogada cada jogador tem 1 de 17 jogadas possíveis. Isto é, uma pedra pode-se mover em 3 direções, 6 pedras podem-se mover em 2 direções e 2 pedras podem-se mover numa única direção. No entanto, tanto o número de pedras como o número de direções possíveis pode variar. Na **Tabela1** é possível visualizar o número de jogadas possíveis mediante o número de pedras em jogo e o número de direções disponíveis.

Tabela 1 – Número de jogadas possíveis.

nº Pedras	nº jogadas possíveis					
#	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18
4	4	8	12	16	20	24
5	5	10	15	20	25	30
6	6	12	18	24	30	36
7	7	14	21	28	35	42
8	8	16	24	32	40	48
9	9	18	27	36	45	54

média de jogadas possíveis	18

Na **Tabela 2** seguinte apresenta-se uma estimativa do número de níveis a percorrer antes de efetuar uma avaliação intermédia do estado do jogo, isto é, utilizar a função de avaliação. Para atingir esta estimativa utilizou-se a média de jogadas possível na **Tabela 1** e calculou-se o número de ramificações para cada nível. De seguida calculou-se o tempo necessário para processar jogadas até um determinado nível de profundidade na árvore de pesquisa. Como é possível verificar, as células da tabela destacadas com core laranja e vermelho revelam tempos de processamento impraticáveis.

Tabela 2 – Estimativa da complexidade temporal.

		Estados processados por segundo			
		1.000	10.000	100.000	1.000.000
nível		Tempo de processamento requerido (sec)			
0	18	0	0	0	0
1	324	0	0	0	0
2	5832	6	1	0	0
3	104976	105	10	1	0
4	1889568	1890	189	19	2
5	34012224	3,E+04	3401	340	34
6	612220032	6,E+05	6,E+04	6122	612
7	1,E+10	1,E+07	1,E+06	1,E+05	11020
8	2,E+11	2,E+08	2,E+07	2,E+06	2,E+05

nível	aceites		média
0	4	4	
1	4	8	
2	4	12	
3	4	16	
4	4	20	
5	3	18	
6	2	14	
7	1	8	
Totais:	26	100	4

Como tal, admitindo os tempos aceitáveis desde do nível 0 até o nível 7 e efetuando uma média ponderada dos níveis aceites para serem processados em tempos exequíveis conclui-se que será possível processar 6 níveis na melhor das hipóteses, 2 na pior das hipóteses e em média até ao nível 4.

Trata-se meramente de uma estimativa matemática sem testes reais podendo ser sujeita a variações significativas, no entanto, essa estimativa prova claramente a necessidade de aplicar uma heurística adequada ao algoritmo *Mini-Max*.

O Projeto

O trabalho foi desenvolvido na linguagem de programação C# com interface gráfica. Através desta interface será possível escolher um de três modos de jogo:

1. Humano contra Humano
2. Humano contra Computador
3. Computador contra Computador

O Computador representa um jogador dotado de uma inteligência artificial. Essa inteligência é implementada através do uso do algoritmo *Mini-Max* com *Cortes Alfa-Beta* apropriado ao jogo em questão.

Sucintamente, o *Mini-Max* com *Cortes Alfa-Beta* é um algoritmo que pesquisa soluções ótimas para um oponente óptimo.

O trabalho foi dividido nas seguintes partes, respeitando a seguinte ordem como fases de implementação:

1. Desenvolvimento do jogo e suas regras
2. Implementação do algoritmo *Mini-Max* com *Cortes Alfa-Beta* e heurísticas subjacentes
3. Desenvolvimento da interface gráfica e interação com o jogador Humano

A representação do conhecimento foi baseada no estado atual do jogo. O estado de um jogo é representado por uma matriz $2 * N - 1$ linhas por $4 * N - 3$ colunas, de forma a facilitar a representação hexagonal do tabuleiro. Para o tabuleiro 5x5, teremos uma matriz 9x17 representada como na **Figura 4**, em que as células coloridas representam as células do tabuleiro hexagonal.

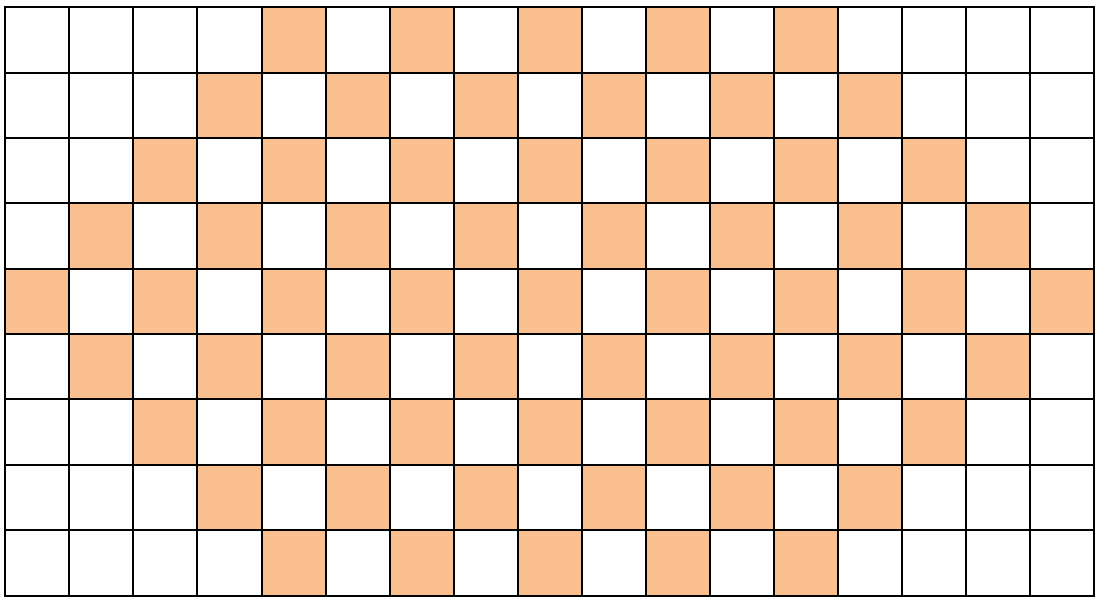


Figura 4 – Estrutura que contém o estado do jogo

Arquitetura do Projeto

Na **Figura 5** é possível visualizar, de forma abstrata, a estrutura do módulo *Game* e respectivas componentes.

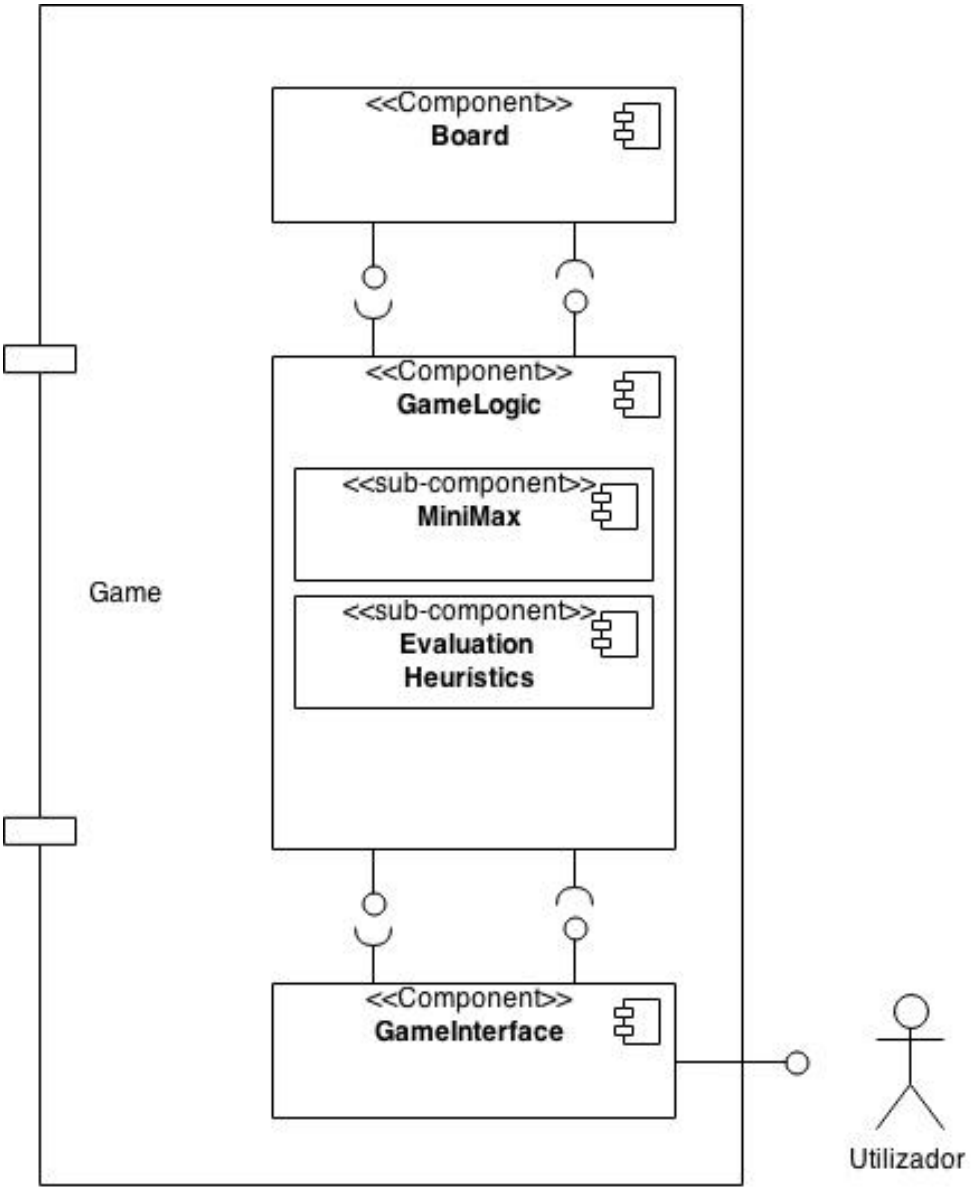


Figura 5 – Modelo de componentes do jogo a implementar.

Metodologia de trabalho

O desenvolvimento do projeto foi efectuado em *Pair-Programming* com uma abordagem *Bottom-Up* começando pela lógica do jogo, incluindo o algoritmo *Mini-Max* simples e *Mini-Max* com *Cortes Alfa-Beta* acabando na interface com o utilizador.

Trabalho Efectuado

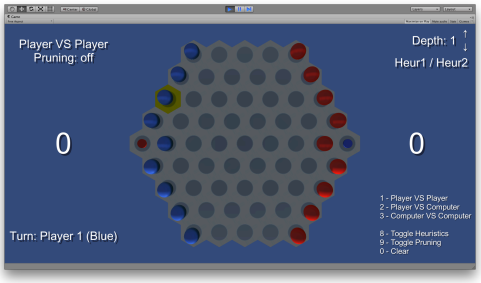
No final deste trabalho foi implementada a lógica de jogo, as heurísticas da função de avaliação, o algoritmo *Mini-Max* simples e o algoritmo *Mini-Max* com *Cortes Alfa-Beta*. Deve-se no entanto referir que o algoritmo *Mini-Max* com *Cortes Alfa-Beta* não está completamente funcional devido ao fato de em alguns casos demorar mais tempo para processar um estado do que com o algoritmo *Mini-Max* simples. Como tal optou-se por apresentar e analisar resultados apenas para o algoritmo *Mini-Max* simples.

O trabalho foi desenvolvido inicialmente em Java, no entanto foi mudado para C# e Unity5 de forma facilitar o uso e criação de ambientes 3D.

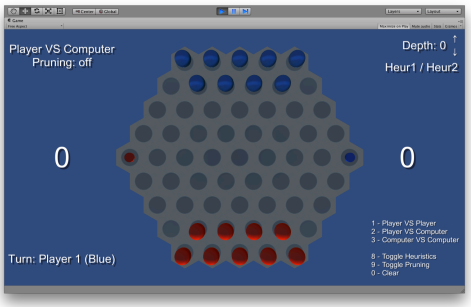
Resultados

No final do projeto obteve-se uma aplicação jogável que permite simular um adversário inteligente. Foram efetuados alguns testes em diferentes situações de jogo (3 estados diferentes) com o algoritmo do Minimax, em profundidades diferentes (entre 1 e 3 inclusivos) de forma a perceber qual a heurística mais competente entre dois jogadores computador com heurísticas diferentes. Estes 3 estados são apresentados nas seguintes imagens:

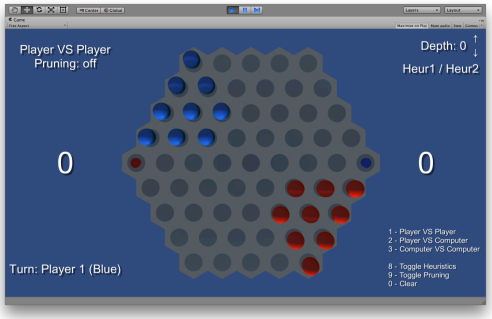
	Estado inicial de jogo 1					
	Profundidade 1		Profundidade 2		Profundidade 3	
Teste	Vencedor	Tempo	Vencedor	Tempo	Vencedor	Tempo
1	2	4,4	1	7,1	1	45
2	2	4,5	2	9,3	2	48
3	1	4,4	1	10,6	2	52
4	2	5,2	2	9,0	1	44
5	2	4,4	1	7,1	2	51
6	2	4,7	2	8,4	2	62
7	1	4,9	1	7,5	2	31
8	2	5,4	2	10,6	2	62
9	1	4,5	2	8,2	2	61
10	1	4,1	2	11,2	2	69
Média	**2**	**4,6**	**2**	**8,9**	**2**	**53**



	Estado inicial de jogo 2					
	Profundidade 1		Profundidade 2		Profundidade 3	
Teste	Vencedor	Tempo	Vencedor	Tempo	Vencedor	Tempo
1	1	1,1	1	1,3	1	32
2	1	0,8	1	1,1	1	12
3	1	1,1	1	1,0	1	24
4	1	0,8	1	1,1	1	13
5	1	0,8	1	1,2	1	13
6	1	0,8	1	1,2	1	11
7	1	0,8	1	1,2	1	7
8	1	0,6	1	0,8	1	15
9	1	0,7	1	1,1	1	13
10	1	0,6	1	1,0	1	27
Média	**1**	**0,8**	**1**	**1,1**	**1**	**17**



Teste	Estado inicial de jogo 3					
	Profundidade 1		Profundidade 2		Profundidade 3	
	Vencedor	Tempo	Vencedor	Tempo	Vencedor	Tempo
1	2	2,7	2	5,6	2	80
2	2	2,9	2	5,0	2	42
3	2	2,8	2	6,0	2	38
4	2	2,8	2	5,7	1	74
5	1	3,2	2	4,3	2	56
6	2	1,8	2	6,1	2	62
7	2	3,1	2	4,7	2	57
8	2	3,0	1	7,3	1	73
9	2	3,4	2	7,0	2	70
10	2	3,5	2	5,2	2	47
Média	**2**	**2,9**	**2**	**5,7**	**2**	**60**



Heurística

Para cada heurística utilizada, foram tomadas em conta as seguintes regras:

- distancia à célula objectivo por parte do jogador
- distancia à célula objectivo por parte do computador
- número de peças em jogo do jogador
- número de peças em jogo do adversário
- número de peças bloqueadas do jogador
- número de peças bloqueadas do adversário
- vitória do jogador
- vitória do adversário

Fórmula utilizada para ambas as heurísiticas:

```
score = (gameState.getWinner()>0) ?
            (
               (pWin         * player_winValue)
            ) :
            (
               (pStones    * player_stoneValue) +
               (eStones     * enemy_stoneValue

               (pBlocked    * player_blockedValue) +
               (eBlocked     * enemy_blockedValue) +

               (pDistance    * player_distanceToGoalValue) +
               (eDistance     * enemy_distanceToGoalValue)
               );
```

Heurística 1

```
player_distanceToGoalValue = +10;
enemy_distanceToGoalValue  = -10;

player_stoneValue     = +50;
enemy_stoneValue     = -50;

player_blockedValue   = +100;
enemy_blockedValue    = -100;

player_winValue      = +10000;
enemy_winValue      = -10000;
```

Heurística 2

player_distanceToGoalValue2 = +20;

enemy_distanceToGoalValue2 = -10;

player_stoneValue2 = +75;

enemy_stoneValue2 = -50;

player_blockedValue2 = +200;

enemy_blockedValue2 = -100;

player_winValue2 = +10000;

enemy_winValue2 = -10000;

Conclusões

Relativamente à complexidade temporal podemos concluir que as nossas estimativas estavam correctas.

Mediante os resultados podemos concluir que ambas as heurísticas são aproximadamente iguais para profundidades inferiores a 3 e que a segunda heurística é superior à primeira para profundidades superiores a 2.

O grupo de trabalho tem clara noção de que a amostra de testes é pequena para efetuar uma conclusão de confiança suficiente. No entanto, dado o tempo disponível foi feito o possível.

Melhoramentos

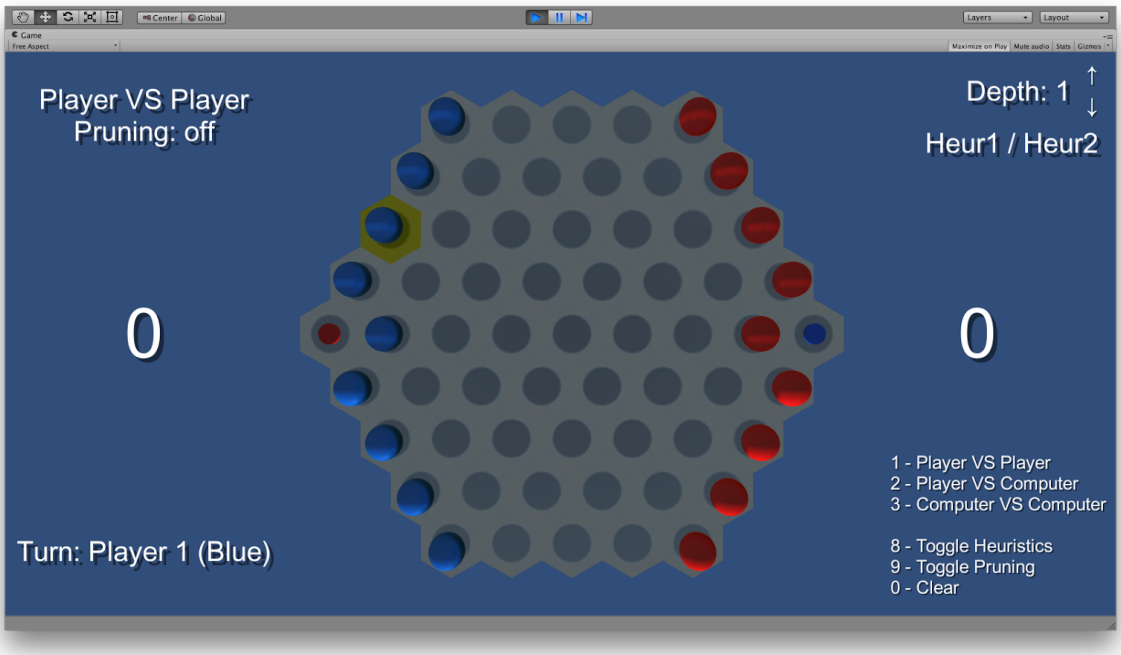
De futuro, a representação do conhecimento pode ser optimizada com o uso de um array simples de tamanho,

$$\sum_{i=0}^{N-1} (2 * (\mathrm{N} + 2\mathrm{i})) \; + 2\mathrm{N} - 1$$

em que para um tabuleiro de 5x5 tem-se as 5 primeiras posições do *array*para a primeira linha, as 7 seguintes para a segunda linha, as 9 seguintes para a terceira linha e por aí em diante, ver exemplo na **Figura 6**.

Figura 6 – Optimização da estrutura que contém o estado do jogo.

Interface gráfica



Na interface gráfica podemos visualizar o estado imediato do jogo, o tipo do jogo, o turno, as heurísticas aplicadas a cada jogador, a profundidade de procura no minimax, o tipo de minimax usado, a pontuação, o tempo da ronda e ainda uma legenda.

De forma a facilitar os ajustes aos testes do jogo e heurísticas foram usados teclados de atalho.

Os atalhos no teclado são:

1 , inicia jogo jogador contra jogador

2 , inicia jogo jogador contra computador

3 , inicia jogo computador contra computador

8 , muda o tipo de combinação de heurística para o grupo de jogadores (H1/H1 ou H1/H2 ou H2/H1 ou H2/H2)

9 , muda o tipo de minimax utilizado, com ou sem cortes alfa-beta

0 , limpa o estado do jogo e ecrã

seta para cima , incrementa o nível de profundidade

seta para baixo , decrementa o nível de profundidade

Também é possível jogar com o rato (quando turno de um jogador não computador) clicando numa peça do jogador cujo turno está em curso e clicando noutra posição para finalizar a jogada dessa peça. No entanto essa ação pode ou não ser permitida.

Recursos

- Referências
  - Russell S., Norvig P., Artificial Intelligence A Modern Approach, Pearson.
  - Artificial Intelligence, CS188.1x, University of Berkeley on BerkeleyX, disponível em: https://courses.edx.org/courses/BerkeleyX/CS188.1x-4/1T2015/courseware/1215d7ff8c304058be1a9abc437ef748/303a2a83d5d04e788fe5bd0c47841bdd/ última consulta em 18 de Março 2015.
  - Sijben P., Aboyne. disponível em: http://www.di.fc.ul.pt/~jpn/gv/aboyne.htm última consulta em 16 de Abril de 2015.
- Software
  - C# e Unity5
  - Java SE, versão 1.8, Oracle
  - Eclipse IDE for Java EE Developers, The Eclipse Foundation.
  - IntelliJ IDEA 14 CE, JetBrains