

Neutreeko

Inteligência Artificial 2020/2021

Turma 03 – Grupo 26

José David Rocha, up201806371

Telmo Costa Botelho, up201806821

Regras do Jogo

Neutreeko é um jogo de tabuleiro de 5x5. Existem dois jogadores: Preto e Branco. A posição inicial das peças é a ilustrada na figura. Tem como objetivo que o jogador coloque as suas três peças numa linha, ortogonal ou diagonalmente ficando as três conectadas.

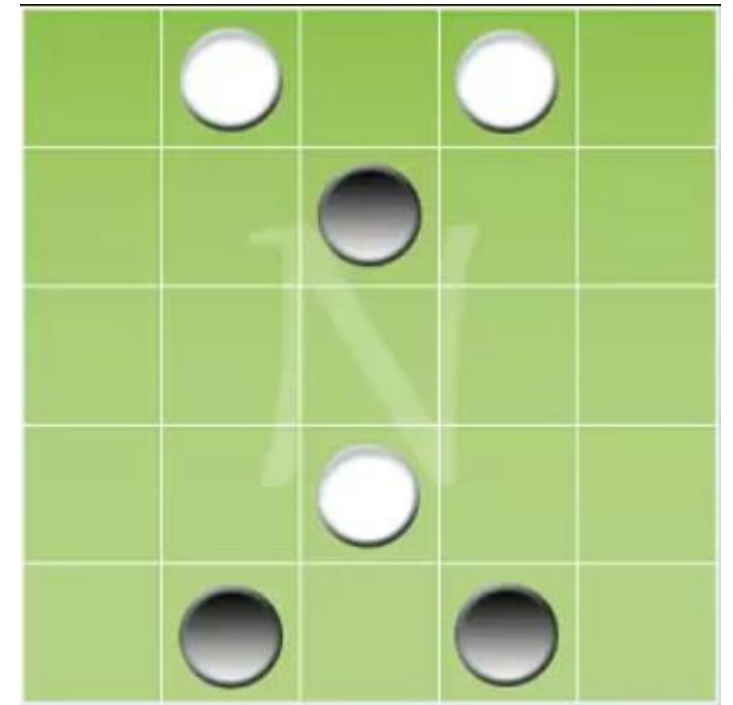
O jogo inicia-se a partir do jogador que controla as peças pretas. Os jogadores, no seu turno, movem alternadamente, cada uma das suas peças.

A peça percorre as células, em linha reta, ortogonal ou diagonalmente até encontrar uma célula ocupada ou a borda do tabuleiro.

O jogo diz-se empatado se a mesma posição ocorre por três vezes seguidas.

Fontes utilizadas:

<http://www.iggamecenter.com/info/pt/neutreeko.html>



Formulação do Problema

Representação do estado

Matriz com o tabuleiro: $B[5,5]$, ou num caso geral $B[N,N]$, preenchido com os valores 0, 1, 2, onde 0 representa uma célula vazia, 1 e 2 peças dos jogadores respetivos e o jogador a efetuar a jogada (Player). Existe também um estado para guardar a ultima jogada efetuada($X1,Y1$) para facilitar a verificação de um vencedor, também haverá um estado para guardar as ultimas jogadas para verificar o acontecimento de um empate.

Estado inicial

$B[5,5]=\{0\}$ except $B[2,1] = 2$, $B[4,1] = 2$, $B[3,4] = 2$, $B[3,2] = 1$, $B[2,5] = 1$, $B[4,5] = 1$.

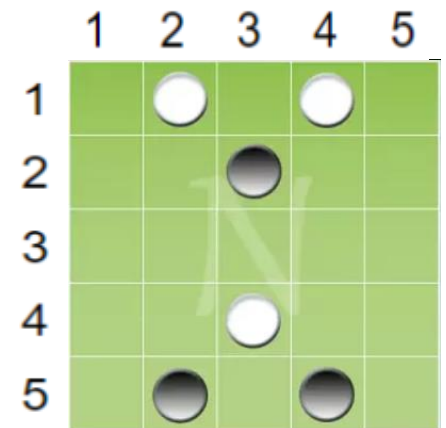
Player = 1 (peças pretas)

Estado Objetivo

//returns 0-empate, 1-Vitória Jogador 1, 2-Vitória Jogador 2, -1-Jogo em progresso

```
def objective_test(State B | Player | Y1 | X1):
```

```
    //Testa 3 em linha em todas as direções a partir de (Y1,X1)
```



Operadores

checkUp(row, col);

checkDown(row, col);

checkRight(row,col);

checkLeft(row, col);

checkUpRight(row, col);

checkUpLeft(row, col);

checkDownRight(row, col);

checkDownLeft(row, col);

Pré-Condições

Nome	Pré-Condições	Efeitos
up	$Y > 1 \wedge (B[X0][Y0-1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
up_right	$X < 5 \wedge Y > 1 \wedge (B[X0+1][Y0-1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
up_left	$X > 1 \wedge Y > 1 \wedge (B[X0-1][Y0-1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
down	$Y < 5 \wedge (B[X0][Y0+1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
down_right	$X < 5 \wedge Y < 5 \wedge (B[X0+1][Y0+1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
down_left	$X > 1 \wedge Y < 5 \wedge (B[X0-1][Y0+1] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
left	$X > 1 \wedge (B[X0-1][Y0] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$
right	$X < 5 \wedge (B[X0+1][Y0] \neq 0)$	$(B[X1][Y1] = \text{Player}) \wedge (B[X0][Y0] = 0)$

Funções Heurísticas

Regra	Pontos
3 Peças da mesma cor em linha	5000
Peça da mesma cor adjacente a outra	5 por ocorrência
Peça da mesma cor na mesma linha/coluna/diagonal	5 por ocorrência

As duas últimas avaliações apenas são feitas caso não existam 3 peças da mesma cor em linha, dado que não faz sentido verificar estes casos para uma situação em que já existe um vencedor da partida.

Existe a possibilidade de utilizar uma função heurística mais simples, que apenas tem em conta se uma *board* tem 3 peças em linhas (ou não), ou uma função heurística mais complexa, que tem em conta os três parâmetros acima descritos.

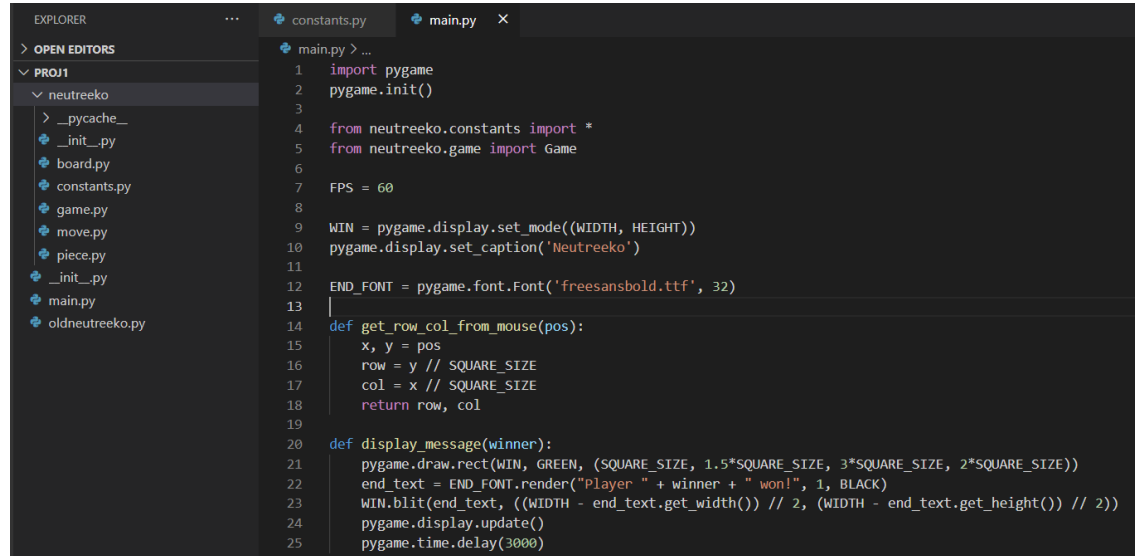
Para além disso, é também adicionada à avaliação da jogada um valor aleatório entre 0 e 5, de modo a poder diferenciar tabuleiros com a mesma avaliação.

Apesar do jogo ser simplista, este envolve um certo nível de tática, que concluímos que passa não só por colocar peças em linha/coluna/diagonal mas também por colocar peças adjacentes umas às outras, incluindo peças adversárias.

```
def evaluation(self):  
    eval = 0  
    player1Pieces = self.getPiecesCoordinates(1)  
    player2Pieces = self.getPiecesCoordinates(2)  
  
    # Verificar 3 em linha  
    if self.check3inARow(player1Pieces):  
        return 5000  
    elif self.check3inARow(player2Pieces):  
        return -5000  
  
    # Verificar se existe 2 em linha/coluna/diagonal  
    # e 2 peças adjacentes  
    if 5000 > eval > -5000:  
        eval += self.check2inLine(player1Pieces)  
        eval -= self.check2inLine(player2Pieces)  
        eval += self.checkSurrounding(player1Pieces)  
        eval -= self.checkSurrounding(player2Pieces)  
  
    return eval + random.randint(0, 5)
```

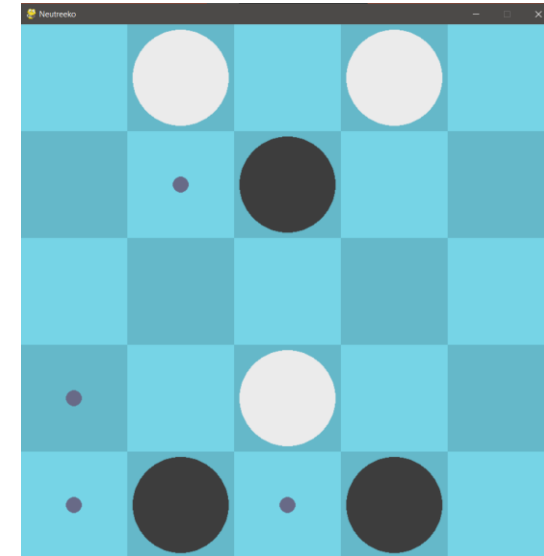
Estado da Implementação até ao Checkpoint

- Linguagem utilizada: Python 3.9
- IDE utilizado: Visual Studio Code
- Estruturas de dados
 - Classe Game – armazena o objeto Board num determinado jogo, posições das peças e contador de hints
 - Classe Board – armazena a Board em si sobre forma de lista de lista
 - Classe Piece – armazena as informações de uma peça, nomeadamente a cor e as suas coordenadas
- Interface do jogo desenvolvida com Pygame 2.0



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane displays the project structure under 'PROJ1' > 'neutreeko'. The files listed are: __pycache__, __init__.py, board.py, constants.py, game.py, move.py, piece.py, __init__.py, main.py, and oldneutreeko.py. The main.py file is open in the editor, showing the following code:

```
1 import pygame
2 pygame.init()
3
4 from neutreeko.constants import *
5 from neutreeko.game import Game
6
7 FPS = 60
8
9 WIN = pygame.display.set_mode((WIDTH, HEIGHT))
10 pygame.display.set_caption('Neutreeko')
11
12 END_FONT = pygame.font.Font('freesansbold.ttf', 32)
13
14 def get_row_col_from_mouse(pos):
15     x, y = pos
16     row = y // SQUARE_SIZE
17     col = x // SQUARE_SIZE
18     return row, col
19
20 def display_message(winner):
21     pygame.draw.rect(WIN, GREEN, (SQUARE_SIZE, 1.5*SQUARE_SIZE, 3*SQUARE_SIZE, 2*SQUARE_SIZE))
22     end_text = END_FONT.render("Player " + winner + " won!", 1, BLACK)
23     WIN.blit(end_text, ((WIDTH - end_text.get_width()) // 2, (HEIGHT - end_text.get_height()) // 2))
24     pygame.display.update()
25     pygame.time.delay(3000)
```



Algoritmos Implementados

- *Minimax*
- *Minimax with alpha/beta pruning*

Estes dois algoritmos foram usados para profundidades de 2, 4 e 5, que correspondem às profundidades de cada um dos *bots* utilizados no modo Jogador vs Computador. No modo Computador vs Computador é possível selecionar a profundidade dos dois *bots* envolvidos na partida.

Nos slides seguintes estão representadas as estatísticas relativas a cada um dos algoritmos implementados. Para tal, foram efetuadas duas jogadas diferentes para cada profundidade, sendo que cada jogada foi repetida 3 vezes, de modo a obter dados mais fidedignos. Para além disso foram tidas em conta as duas funções heurísticas implementadas, para avaliar qual o grau de influência destas na eficiência temporal de cada jogada.

Para além disso, encontram-se em Anexo os gráficos gerados, que servem como forma de melhorar a interpretação dos resultados obtidos e consequentes conclusões retiradas.

Algoritmo	Profundidade	Nós percorridos		Tempo (s)		Média Nós Percorridos		Tempo Médio (s)	
		Heur 1	Heur 2	Heur 1	Heur 2	Heur 1	Heur 2	Heur 1	Heur 2
Minimax	2	264	180	0.0268	0.0204	243.3	180	0.2523	0.0207
		233	180	0.0233	0.0209				
		233	180	0.0256	0.0208				
		215	215	0.026	0.0243	215	215	0.0257	0.0242
		215	215	0.0289	0.0243				
		215	215	0.0222	0.0240				
	4	3593	5522	0.9462	1.1318	3638	5499.7	0.8353	1.1574
		3604	5660	0.7538	1.2414				
		3717	5317	0.8058	1.0991				
		6800	10703	1.6814	2.4559	5842	10370	1.4137	2.5021
		5260	10344	1.2756	2.5968				
		5466	10063	1.284	2.4536				
	5	11749	18812	6.2939	7.6111	12635.6	19167	6.5005	7.4793
		12705	18782	6.5366	7.4153				
		13453	19907	6.6771	7.4118				
		40108	40154	13.7265	12.3533	38760.3	42543	14.0072	11.4833
		40848	40257	15.7559	10.7142				
		35325	42718	12.5391	11.3824				

Algoritmo	Profundidade	Nós percorridos		Tempo (s)		Média Nós Percorridos		Tempo Médio (s)	
		Heur 1	Heur 2	Heur 1	Heur 2	Heur 1	Heur 2	Heur 1	Heur 2
<i>Minimax With Alpha/Beta Cuts</i>	2	47	71	0.0231	0.0227	50.3	71	0.0256	0.0229
		60	71	0.0272	0.0229				
		44	71	0.0265	0.0232				
		96	111	0.0279	0.0209	102	110.3	0.0283	0.0201
		109	97	0.0296	0.0201				
		101	123	0.0275	0.0194				
	4	844	1928	0.2772	0.7218	953.7	1970.3	0.3529	0.7086
		792	2097	0.2538	0.7424				
		1225	1886	0.5278	0.6616				
		1427	3146	0.6708	1.3595	1570	3168.3	0.6673	1.3924
		1469	3174	0.6254	1.3708				
		1814	3185	0.7058	1.4469				
	5	3624	13151	1.1649	2.7493	3771.7	13172.6	1.2266	2.8092
		4135	13110	1.3069	2.7226				
		3556	13257	1.2080	2.9556				
		5334	17642	1.7905	3.7276	6097.3	16967.3	2.1837	3.6088
		5735	18089	2.2038	3.8569				
		7223	15171	2.5567	3.2418				

Conclusões

Com base nas tabelas dos *slides* anteriores, é possível concluir que:

- O tempo de processamento do algoritmo Minimax é exponencial relativamente à profundidade da árvore gerada e ao *branching factor* associada à mesma, mesmo que sejam utilizados *cuts*.
 - É notória a melhoria em termos de eficiência temporal bem como em número de nós processados através do uso de *cuts*, sendo que quanto maior a profundidade usada, melhor o desempenho destes. Por exemplo, para uma profundidade de 5, é possível verificar que houve melhorias de 14 para 2 segundos de processamento. Para além disso, a ordenação prévia dos nós por ordem decrescente de avaliação contribui para uma melhoria na eficiência do algoritmo.
 - Contudo, o uso de *cuts* não revela grandes melhorias temporais para profundidades baixas, como é visível nos gráficos relativos a profundidade 2, apesar do número de nós processados ser ligeiramente menor.
- O uso de funções heurísticas mais complexas influencia o desempenho global do algoritmo.
 - À semelhança dos *cuts*, a utilização de uma função heurística complexa não prejudica o desempenho do algoritmo para profundidades baixas.
 - Contudo, para profundidades mais elevadas, é notório que o desempenho do algoritmo se torna pior, chegando mesmo a duplicar o tempo de processamento de uma jogada bem como o número de nós processados.
- Sendo o jogo bastante simplista nota-se que profundidades baixas já são capazes de jogar o jogo bastante bem.

Referências/Materiais

- [GitLab com implementação em Java de Neutreeko](#)
- Slides das aulas teóricas fornecidas pelos docentes
- [Pygame](#)