Alquerque

Entrega Final

Trabalho Realizado Por:

Miguel Proença Fernandes Ramalho Amaro, up202106985

António Maria Araújo Pinto dos Santos, up202105587

Índice

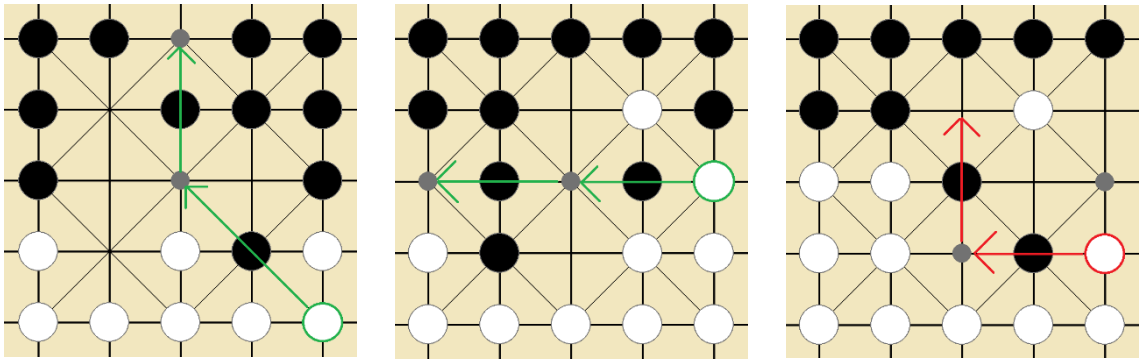
Trabalho a Desenvolver

O trabalho escolhido para desenvolvimento foi a opção 2B (Alquerque). Este jogo, oriundo do Médio Oriente em meados do século X, é tido como a origem do (mais tarde desenvolvido) jogo das damas.

O alquerque é jogado por dois jogadores num tabuleiro de dimensão 5x5, onde cada jogador assume 12 peças de uma cor (preta ou branca) e tem como objetivo eliminar as peças do adversário. Embora as regras tenham sofrido várias alterações ao longo dos anos, e devido a um não consenso sobre as mesmas, iremos utilizar as regras originais, de seguida enunciadas:

- Cada jogador coloca as suas peças nas duas filas mais próximas de si, e nas duas células mais à sua direita da fila do meio;
- Pode-se movimentar uma peça em qualquer direção adjacente (exceto para trás), desde que esta esteja vazia e possua uma linha a conectá-la à célula em que a peça se encontra;
- Pode-se movimentar uma peça por cima de uma peça do adversário, desde que a célula imediatamente seguinte esteja vazia, e estas posições estejam conectadas por uma linha;
- O movimento por cima de uma peça do adversário leva à eliminação da peça do adversário;
- É possível fazer eliminações contínuas com uma só peça numa só jogada, desde que as condições anteriormente mencionadas estejam reunidas, e só estejam envolvidos movimentos num só sentido.

Exemplos de movimentos válidos (verde) e inválidos (vermelho):

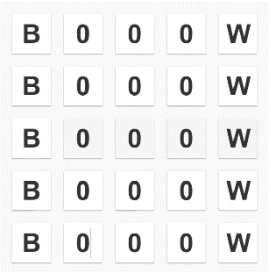


Problema de Busca

- ## State Representation

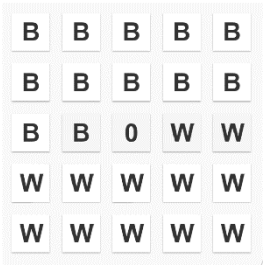
O tabuleiro pode ser representado por uma matriz de dimensão 5x5. Cada célula poderá estar desocupada (0), ocupado por uma peça branca (W) ou ocupada por uma peça preta (B).

Por exemplo, um tabuleiro com a coluna esquerda preenchida com peças pretas (B), a coluna direita preenchida com peças brancas (W), e as restantes células desocupadas (0), representa-se da forma indicada na imagem à direita.



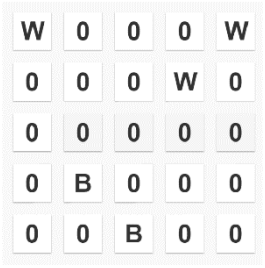
- ## Initial State

O tabuleiro terá as duas filas mais próximas de cada jogador e os lugares mais à direita de cada jogador da fila do meio ocupadas com peças da respetiva cor. Assim, temos o estado inicial do tabuleiro na imagem à direita.



- ## Objective Test

O jogo termina quando um jogador elimina todas as peças do adversário, sobrando só peças de uma cor no tabuleiro. É possível empatar, por exemplo, no caso de as únicas peças restantes estarem no canto do adversário, só se podendo movimentar para os lados. Uma instância deste caso está representada na imagem à direita.



- **Evaluation Functions**

Para criar um programa capaz de jogar alquerque, será implementado um algoritmo MiniMax com Alfa-Beta Cuts. As diferentes dificuldades serão obtidas através do uso de diferentes profundidades de pesquisa no algoritmo, sendo o programa melhor no jogo quanto maior for a profundidade.

- **Operators**

Sendo as coordenadas de uma peça, antes de fazer a jogada, representadas por m[i][j]; X o tipo de peça correspondente à do jogador a fazer a jogada; Y o tipo de peça do adversário; e 0 uma célula vazia, temos os seguintes operadores:

Nome	Precondições	Efeitos	Custo
1F	m[i][j]=X ∧ m[i-1][j]=0	m[i][j]=0 ∧ m[i-1][j]=X	1
1T	m[i][j]=X ∧ m[i+1][j]=0	m[i][j]=0 ∧ m[i+1][j]=X	1
1D	m[i][j]=X ∧ m[i][j+1]=0	m[i][j]=0 ∧ m[i][j+1]=X	1
1E	m[i][j]=X ∧ m[i][j-1]=0	m[i][j]=0 ∧ m[i][j-1]=X	1
1FD	m[i][j]=X ∧ m[i-1][j+1]=0	m[i][j]=0 ∧ m[i-1][j+1]=X	1
1FE	m[i][j]=X ∧ m[i-1][j-1]=0	m[i][j]=0 ∧ m[i-1][j-1]=X	1
1TD	m[i][j]=X ∧ m[i+1][j+1]=0	m[i][j]=0 ∧ m[i+1][j+1]=X	1
1TE	m[i][j]=X ∧ m[i+1][j-1]=0	m[i][j]=0 ∧ m[i+1][j-1]=X	1
2F	m[i][j]=X ∧ m[i-1][j]=Y ∧ m[i-2][j]=0	m[i][j]=0 ∧ m[i-1][j]=0 ∧ m[i-2][j]=X	1
2T	m[i][j]=X ∧ m[i+1][j]=Y ∧ m[i+2][j]=0	m[i][j]=0 ∧ m[i+1][j]=0 ∧ m[i+2][j]=X	1
2D	m[i][j]=X ∧ m[i][j+1]=Y ∧ m[i][j+2]=0	m[i][j]=0 ∧ m[i][j+1]=0 ∧ m[i][j+2]=X	1
2E	m[i][j]=X ∧ m[i][j-1]=Y ∧ m[i][j-2]=0	m[i][j]=0 ∧ m[i][j-1]=0 ∧ m[i][j-2]=X	1
2FD	m[i][j]=X ∧ m[i-1][j+1]=Y ∧ m[i-2][j+2]=0	m[i][j]=0 ∧ m[i-1][j+1]=0 ∧ m[i-2][j+2]=X	1
2FE	m[i][j]=X ∧ m[i-1][j-1]=Y ∧ m[i-2][j-2]=0	m[i][j]=0 ∧ m[i-1][j-1]=0 ∧ m[i-2][j-2]=X	1
2TD	m[i][j]=X ∧ m[i+1][j+1]=Y ∧ m[i+2][j+2]=0	m[i][j]=0 ∧ m[i+1][j+1]=0 ∧ m[i+2][j+2]=X	1
2TE	m[i][j]=X ∧ m[i+1][j-1]=Y ∧ m[i+2][j-2]=0	m[i][j]=0 ∧ m[i+1][j-1]=0 ∧ m[i+2][j-2]=X	1

Implementação

A implementação deste trabalho foi realizada no IDE Visual Studio Code 1.76.1, no sistema operativo Linux Mint Cinammon 21.1. Foi utilizada a linguagem de programação Python, à qual associamos o uso do Pygame, que permitiu o desenvolvimento de uma interface gráfica.

O programa foi dividido em várias pastas (em sublinhado) e ficheiros (**em negrito)**, de modo a facilitar a navegação e a interpretação dos mesmos. Dentro da pasta "jogo_final" temos:

- Ficheiro **main.py**, pelo qual todos os outros ficheiros são acedidos, iniciando o tabuleiro, recebendo posições e inputs do jogador;
- Pasta alquerque, que contem os ficheiros responsáveis pelo funcionamento do jogo. São estes:
    - Ficheiro **board.py**, responsável por desenhar o tabuleiro, colocar as peças no tabuleiro, definir os movimentos legais, …;
    - Ficheiro **game.py**, responsável por definir as regras do jogo, isto é, eliminações de peças, fim do jogo, mudanças de turno, …;
    - Ficheiro **piece.py**, responsável por desenhar as peças e permitir o seu movimento;
    - Ficheiro **constants.py**, responsável por guardar as constantes mais importantes, permitindo uma rápida alteração das mesmas e facilitando a interpretação dos valores nos outros ficheiros;
- Pasta minimax, que contém o ficheiro **algorithm.py**, responsável pela definição do algoritmo Minimax usado nos modos que envolvem um ou dois jogadores não humanos.

Abordagem

O ficheiro **main.py**, pelo qual o jogo é iniciado (na consola, $python main.py), começa por iniciar a função "*inputs*", que recebe o modo de jogo escolhido pelo jogador e, dependendo das escolhas feitas, inicia uma das funções "mainHvH" (Humano vs. Humano), "mainHvP" (Humano vs. PC) ou "mainPvP" (PC vs. PC), com os respetivos atributos escolhidos.

O local em que se clica é obtido pela função "get_row_col_from_mouse", que é chamada nas funções "main" anteriormente mencionadas. Estas funções vão chamando outras funções, de outros ficheiros, de modo a permitir jogar alquerque. Entre todas, as mais importantes são:

- "draw_grid"; desenha o tabuleiro (chamado em"draw").
- "evaluate"; é a função de avaliação usada no Minimax.
- "get_all_pieces"; obtém as peças da cor do jogador.
- "move"; se o movimento escolhido for legal, executa-o.
- "create_board"; coloca as peças no tabuleiro.
- "remove"; remove peças eliminadas e altera o contador destas.
- "winner"; define se alguém venceu o jogo;
- "get_valid_moves"; deteta os movimentos legais através das funções "_traverse_left", "_traverse_right", "_vertical" e "_horizontal", e guarda-os num dicionário *moves* (utilizamos um dicionário em vez de lista, pois não necessitamos de ordenar os movimentos, tornando o programa mais rápido).
- "draw_valid_moves"; utiliza *moves* de "get_valid_moves" para desenhar os movimentos legais no tabuleiro.
- …

Algoritmos Implementados

Na pasta minimax, o ficheiro **algorithm.py** contem o algoritmo utilizado para jogar alquerque. É composto por 3 funções principais: “minimax”, “simulate_move” e “get_all_moves”.

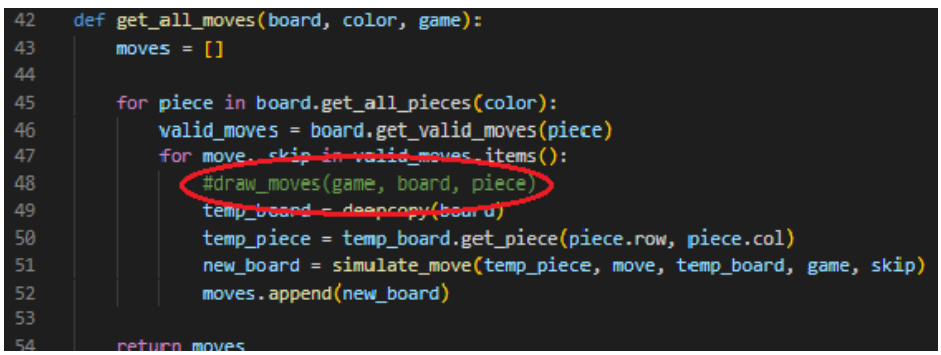
A função “minimax” recebe o estado do tabuleiro (*position*), a profundidade de pesquisa desejada (*depth*), o jogador que pretende maximizar a função de avaliação (*max_player*, neste caso são as peças pretas, pois a função de avaliação equivale a #PeçasPretas menos #PeçasBrancas) e a janela (*game*).

Se a profundidade for 0 ou se houver um vencedor, não se dá nenhum movimento. Através da recursão, esta função é repetidamente chamada (com alteração do turno do jogador, sabido pelo argumento *max_player*) até ser atingida a *depth* desejada, e retornado o valor adequado.

A função “simulate_move”, como o nome indica, simula a alteração do tabuleiro de acordo com o movimento fornecido.

A função “get_all_moves” é chamada por “minimax”, e obtém todos os movimentos possíveis a partir do tabuleiro fornecido, simulando-os através do chamamento de “simulate_move”.

Nesta função pode retirar o cardinal da linha 48 (assinalada na imagem em baixo), de modo a revelar todos os movimentos que o computador pondera realizar antes de executar o movimento final.



Resultados e Notas

Após testar todas as combinações possíveis do modo PC vs. PC (4 dificuldades das peças brancas por cada uma das 4 dificuldades das peças brancas, total de 16 combinações), foi possível extrair várias conclusões. Por exemplo:

- Mesmo com uma discrepância no nível da dificuldade dos 2 programas, é possível, por vezes, ocorrer um empate (por exemplo, quando as peças pretas têm nível de dificuldade 3 as peças brancas têm nível de dificuldade 1);
- Por vezes, é possível ganhar o programa com nível de dificuldade menor (por exemplo, quando as peças pretas têm nível de dificuldade 1 e as peças brancas têm nível de dificuldade, ganham as peças pretas);
- Quanto maior for o nível de dificuldade, maior será o tempo necessário para executar a jogada (pois o número de estados explorados é exponencial);
- …

É importante notar que foi, também, implementada uma versão da função “minimax” com alpha-beta pruning. De qualquer modo, o uso desta função resultava em alguns erros alheios, não sendo consistente, pelo que se preferiu usar o Minimax sem alpha-beta pruning (todavia, esta versão encontra-se preservada e pode ser apresentada).

Referências Bibliográficas

https://play.google.com/store/apps/details?id=com.noApp.alquerque

https://en.wikipedia.org/wiki/Alquerque

https://boardgamegeek.com/boardgame/11464/alquerque

https://github.com/Tartori/gametheory-alquerque

https://archive.codecup.nl/2008/examples/alquerque_py.html

https://rubysash.com/programming/perl/perl-script-to-make-alquerque-game-boards-in-svg/

https://youtube.com/playlist?list=PLzMcBGfZo4-lkJr3sqpikNyVzbNZLRiT3