U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

L.EIC 2022/2023
Inteligência Artificial

Assignment No.1

Adversarial Search Methods for Games

3LEIC - G33
António Ferreira – up202004735@edu.fe.up.pt
Hugo Gomes – up202004343@edu.fe.up.pt
João Moreira – up202003550@edu.fe.up.pt

Especificação do Trabalho

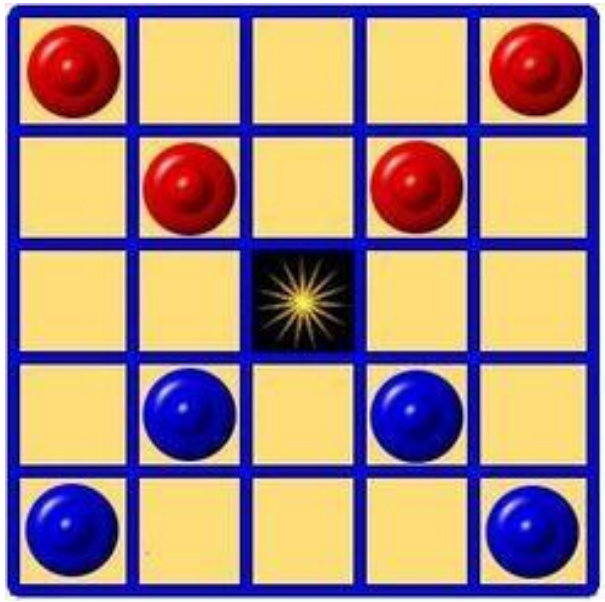
Para este projeto, o objetivo é desenvolver uma adaptação do jogo “Black Hole Escape”, no sentido de estudar Adversarial Search Methods para jogos de 2 jogadores.

Quanto ao jogo, este é jogado num tabuleiro 5x5, sendo que o espaço central é considerado um “black hole”. O principal objetivo do jogo é tentar ser o primeiro jogador a colocar duas das suas peças no centro.

Cada jogador começa com 4 peças, caracterizadas como sendo naves espaciais de cor específica a cada indivíduo. Cada peça pode mover-se na vertical ou na horizontal, pelo que os movimentos diagonais são proibidos. Segundo o contexto do jogo, estas naves encontram-se numa dimensão na qual os seus travões não estão a funcionar. Desta forma, sempre que uma peça se mexe, o seu movimento só para quando atingir uma outra peça ou uma fronteira do tabuleiro.

Como já foi referido, ambos os jogadores têm de tentar colocar as suas peças no “black hole” central, pelo que se uma nave calhar nesta posição ,será transportada para uma nova dimensão e, consequentemente, removida do tabuleiro. É de notar que as naves podem passar sobre esta posição central, sendo que só serão removidas quando calharem neste quadrado.

O jogador que for o primeiro a salvar duas das suas naves desta dimensão é declarado o vencedor.



(fig.1 – Black Hole Escape (estado inicial)

https://boardgamegeek.com/boardgame/64229/black-hole-escape

Formulação do problema como um ‘Search Problem’

State Representation – Uma matriz (vetor de duas dimensões) de 5 linhas e 5 colunas, onde nas posições ocupadas pelas peças do primeiro jogador encontra-se ‘1’ e nas posições das naves do segundo jogador um ‘2’. O ‘black hole’ encontra-se nas coordenadas (2,2) e é representado por um ‘3’.

Os restantes espaços vazios são representados por ‘0’.

Initial State – O estado inicial deste problema consiste na posição inicial das naves no tabuleiro.

Goal/Objective Test – Verifica se um dos dois jogadores conseguiu com que duas das suas peças calhassem no ‘black hole’.

Heuristics Function – O principal objetivo desta função é estimar o número mínimo de jogadas necessárias para se atingir o estado final. Sendo assim, pode-se determinar a ‘**Manhattan Distance**’ e a **‘Euclidean Distance’** entre as naves restantes de um jogador e o ‘black hole’. Também é de destacar a possibilidade de escolher uma **peça random** e que esta se movimente numa **direção aleatória**. E, por fim, também será possível calcular a **distância mínima entre naves do adversário**, no sentido de evitar movimentos que permitam ao outro jogador fazer um ponto, por exemplo.

Operators –

- **Move(piece, direction) –** Este operador permite um jogador movimentar uma das suas peças (piece) numa determinada direção (direction)
- **Land(piece) –** Este operador permite que um jogador remova a sua peça do tabuleiro, pois está no ‘black hole’.

(*)

```
[[1,0,0,0,1],
 [0,1,0,1,0],
 [0,0,3,0,0],
 [0,2,0,2,0],
 [2,0,0,0,2]]
```

(fig.2 – Representação matricial do estado inicial.

(*)

Move –

- Preconditions – Para a jogada ser válida, a peça tem que se movimentar numa direção válida (esquerda, cima, direita ou baixo) e que não esteja diretamente bloqueada por uma outra peça ou fronteira.
- Effects – A peça movimentou-se para um novo espaço e a vez de jogar é passada para o próximo jogador.
- Cost – 1

Land –

- Preconditions – A nave tem de estar no ‘black hole’.
- Effects – A peça é removida do jogo.
- Cost - 1

Implemented Work

- **Programming Language** – Python (pygame - https://www.pygame.org/docs/)
- **Development Environment** – VSCode
- Data Structures –
  - **Piece Class**
    - changeColor() *//changes its color*
    - draw() *//draws the sprite on the screen*
    - setDrawX() *//sets the x coordinate of the sprite*
    - setDrawY() *//sets the x coordinate of the sprite*
    - checkCollision() *//checks if a collision happens when the piece moves*
    - checkBlackHole() *//checks if the piece reached the black hole*
    - move() *//moves the piece according to a given direction*
    - isClicked() *//checks if the piece was clicked*
    - availableMoves() *//returns the coordinates of the available moves to a piece*
  - **Player Class**
    - movePiece() *//moves a chosen piece*
    - selectPiece() *//selects one of the player's pieces*
    - resetPieces() *//resets the player's pieces after their turn*
    - removePiece() *//removes a player piece after it landed on the black hole*
    - availableMoves() *//returns a list with the available moves of all the player's pieces*
  - **Board Class**
    - drawInit() *//draws the initial state of the board*
    - draw() *//updates the board after a player's turn*
    - updateMatrix() *//updates the board's matrix*
    - removePiece() *//removes a piece from the board's piece list and matrix*

Implemented Work

- **Computer Class**
  - movePiece() *//moves a chosen piece*
  - selectPiece() *//selects one of the computer's pieces*
  - resetPieces() *//resets the computer's pieces after their turn*
  - removePiece() *//removes a computer piece after it landed on the black hole*
  - availableMoves() *//returns a list with the available moves of all the computer's pieces*
- **Game Class**
  - play() *//plays the game*
  - checkWinner() *//checks if a player has won the game*
  - winnerScreen() *//shows the final screen*

- **Implemented features –**
  - Mover as peças
  - Desenhar as peças
  - Desenhar e atualizar o tabuleiro
  - Selecionar peça para jogar
  - Salvar peças
  - Obter uma lista de possíveis jogadas para cada peça
  - Verificar colisões
  - Ganhar um jogo
  - Mostrar ecrã final
  - Obter a duração de um jogo (em segundos)
  - Player versus Computer (computador efetua jogadas aleatórias)