Resolução do jogo Blockage utilizando o método de pesquisa MiniMax alfa-beta e variantes em linguagem Java (Tema A3 Grupo 5)

1st Eduardo Silva (up201603135)
MIEIC
FEUP
Porto, Portugal
up201603135@fe.up.pt

2nd Mariana Costa (up201604414)
MIEIC
FEUP
Porto, Portugal
up201604414@fe.up.pt

3rd Tiago Castro (up201606186)
MIEIC
FEUP
Porto, Portugal
up201606186@fe.up.pt

***Resumo*—O presente artigo tem como principal objetivo delinear os contornos de desenvolvimento da resolução do jogo Blockage utilizando o método de pesquisa MiniMax alfa-beta em linguagem Java. Será realizada uma descrição sucinta do jogo em questão, seguida da formulação do mesmo como problema de pesquisa bem como a exposição de outros trabalhos semelhantes considerados úteis e uma breve conclusão a destacar as perspetivas de desenvolvimento.**

***Index Terms*—Inteligência Artificial, Jogo, Bloqueio, MiniMax, Pesquisa com adversários**

I. Introdução

O trabalho aqui exposto insere-se no âmbito da unidade curricular de Inteligência Artificial do MIEIC. É proposta a resolução do Blockage, um jogo competitivo, utilizando métodos de pesquisa MiniMax alfa-beta e variantes em linguagem Java. Este deverá disponibilizar um modo de visualização textual ou gráfica, de forma a mostrar com clareza a evolução do tabuleiro e possibilitar a comunicação entre a máquina e o utilizador/jogador. Deverá igualmente permitir um modo de jogo no qual o computador conduz o jogo autonomamente, com recurso a um método e configuração selecionados previamente pelo utilizador.

II. Descrição do problema

Blockage [1] é um jogo competitivo no qual o objetivo principal consiste em percorrer o tabuleiro de jogo, alcançando o lado cuja cor corresponde à cor do jogador em questão, sendo o vencedor aquele que cumprir este objetivo em primeiro lugar. Em cada turno, cada jogador pode optar por avançar uma casa ou colocar uma barreira.

III. Formulação do problema

A. Representação do estado

Estados representados por uma matriz 2D de carateres constituída por quatro elementos principais:

- Posição na qual pode ser colocada uma barreira
- Posição para a qual o jogador se pode deslocar
- Barreira
- Jogador

De notar que cada barreira ocupa três posições seguidas na matriz, vertical ou horizontalmente. As posições de fronteira para as quais os jogadores se podem deslocar (ou seja, os limites do tabuleiro) serão demarcadas das demais através da indicação da sua cor, de forma a que cada jogador saiba para que lado se deslocar.

Na figura seguinte encontra-se um exemplo de uma eventual representação textual do estado de jogo, cujos elementos se encontram enumerados e descritos de seguida:

- **'A'/'V'**: Jogador azul/verde
- **'a'/'v'**: Fronteira azul/verde
- **'B'**: Barreira
- **'_'**: Posição para a qual o jogador se pode deslocar
- **Espaço vazio**: Posição na qual pode ser colocada uma barreira

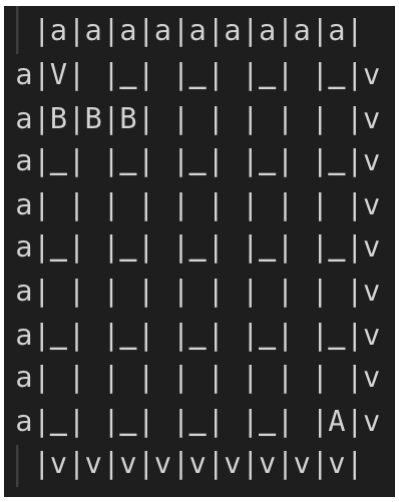

Figura 1. Possível representação textual de um tabuleiro de jogo

B. Estado inicial

Jogadores dispostos em lados opostos à fronteira da sua cor.

C. Teste terminal

Para que um jogador seja considerado vencedor, é necessário estar numa posição adjacente à fronteira da mesma cor que a sua.

D. Operadores

- *Mover jogador (esquerda, direita, cima, baixo)*
  - **Pré-condições**: A posição de destino deverá estar demarcada pelo carater '_', dentro dos limites do tabuleiro e a uma posição de distância (2 celulas) da posição atual.
  - **Efeitos**: O jogador move-se uma posição na direção indicada.
- *Colocar barreira*
  - **Pré-condições**: A posição de destino deverá permitir a criação de uma barreira que ocupe 3 posições (horizontal ou verticalmente) sendo que a primeira e a última posição da barreira deverão estar adjacentes a uma celula '_', também elas igualmente válidas.
  - **Efeitos**: É criada uma barreira ocupando 3 espaços previamente vazios na posição e direção indicadas.

E. Função utilidade

Para calcular a utilidade de cada estado usaremos uma função que retorne um valor que representa não só a distância do próprio jogador à barreira objetivo como também o valor (ou a soma no caso de vários jogadores) da distância do jogador adversário à barreira da sua cor.

IV. Trabalho relacionado

Como o jogo que aqui se propõe realizar é bastante específico, a nossa pesquisa centrou-se mais no algoritmo MiniMax em si. Também nos focámos mais em fontes que tivessem Java como linguagem principal de modo a facilitar a possível incorporação de código no nosso trabalho. Como tal, foi encontrado um repositório [2] com uma implementação do algoritmo MiniMax com cortes Alpha e Beta do jogo do galo (note-se que há bastantes implementações deste jogo com MiniMax em vários repositórios no entanto esta pareceu-nos uma das melhores). Além deste, foram também encontrados diversos websites sobre o mesmo tópico que considerámos relevantes [3], [4].

V. Conclusões e perspetivas de desenvolvimento

Após análise do trabalho a desenvolver concluímos que o método de estruturação do projeto previamente apresentado é o melhor para obter um bom resultado no mesmo. Assim, implementaremos, em Java, os operadores designados procedendo seguidamente a testar os métodos de pesquisa de forma a otimizar a obtenção de uma resolução do problema.

Referências

[1] https://play.google.com/store/apps/details?id=br.com.nespolo.activity
[2] https://github.com/LazoCoder/Tic-Tac-Toe
[3] https://www.e4developer.com/2018/09/23/implementing-minimax-algorithm-in-java/
[4] https://tutorialedge.net/artificial-intelligence/min-max-algorithm-in-java/