

README

1. Jogo

Jeson Mor_3

- Filipe de Moraes Teixeira Pinto (up201907747)
- Juan Bellon Lopez (up201908142)

2. Instalação e execução

Windows

- Executar `SicStus Prolog`
- `File -> Consult(...)` -> Selecionar ficheiro `play.pl`
- Na consola do SicStus: `play.`

Linux

- Executar `SicStus Prolog`
- `File -> Consult(...)` -> Selecionar ficheiro `play.pl`
- Na consola do SicStus: `play.`

3. Descrição do Jogo:

Jeson Mor (Nove Cavalos em Português) é um jogo de estratégia de origem Mongol jogado num tabuleiro 9x9. Cada um dos jogadores começa o jogo com 9 cavalos de xadrez (que fazem os mesmos movimentos em L) alinhados na primeira linha do seu lado do Tabuleiro. Os jogadores deveram estar no lado oposto um do outro. O objetivo do jogo é chegar ao centro do Tabuleiro (E5) e conseguir sair ou capturar todos os cavalos do adversário. Os jogadores jogam alternadamente tal como iniciam o jogo alternadamente. *Fonte:* <https://en-academic.com/dic.nsf/enwiki/11675050>

4. Lógica do Jogo:

4.1 Representação interna do estado do jogo:

Tabuleiro

O nosso tabuleiro é representado a partir de uma lista com sublistas, sendo cada sublista uma linha. É inicializado de forma recursiva para preencher a primeira e última linha com os respectivos elementos (`1` de um lado e `-1` do outro), o número `3` no centro e o resto é representado com o número `0`. Também tem um tamanho fixo de `9 x 9`. Cada elemento, durante o jogo, pode ter 1 de 4 valores possíveis no código:

- `0` -> representa uma posição vazia (No Tabuleiro =)
- `1` -> representa uma posição com uma peça pertencente ao jogador 1 (Tabuleiro = `0`)
- `-1` -> representa uma posição com uma peça pertencente ao jogador 2 (Tabuleiro = `X`)
- `3` -> representa a posição do meio (Tabuleiro = `?`)

Código utilizado para a representação de elementos

```
character(0, ' ').
character(-1, 'X').
character(1, 'O').
character(3, '?').
```

Estados de Jogo

```
%Inicial
[ [ 1,1,1,1,1,1,1,1,1 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,0,3,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ -1,-1,-1,-1,-1,-1,-1,-1 ] ]

%Intermedio
[ [ 1,1,0,1,0,1,1,0,1 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,1,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,1,0,3,1,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,-1,0,0,0,-1,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ -1,-1,0,-1,-1,0,0,0 ] ]

%Final
[ [ 1,1,0,1,0,1,0,0,1 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,0,1,0,0,0,0,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ 0,0,1,0,3,1,0,0,0 ],
  [ 0,0,*1*,0,0,0,0,0,0 ], %peça *1* conseguiu sair do meio logo o jogo acaba
  [ 0,0,0,-1,0,0,0,-1,0 ],
  [ 0,0,0,0,0,0,0,0,0 ],
  [ -1,-1,0,-1,-1,0,0,0 ] ]
```

Jogadores

No nosso existem dois tipos:

- **Player** - Para quando for a vez de um jogador humano

- **Easy** - Para quando for a vez do Computador, em dificuldade fácil (joga sempre para posições legais de maneira aleatória).

4.2 Visualização do estado de jogo:

Percorrer os Menus

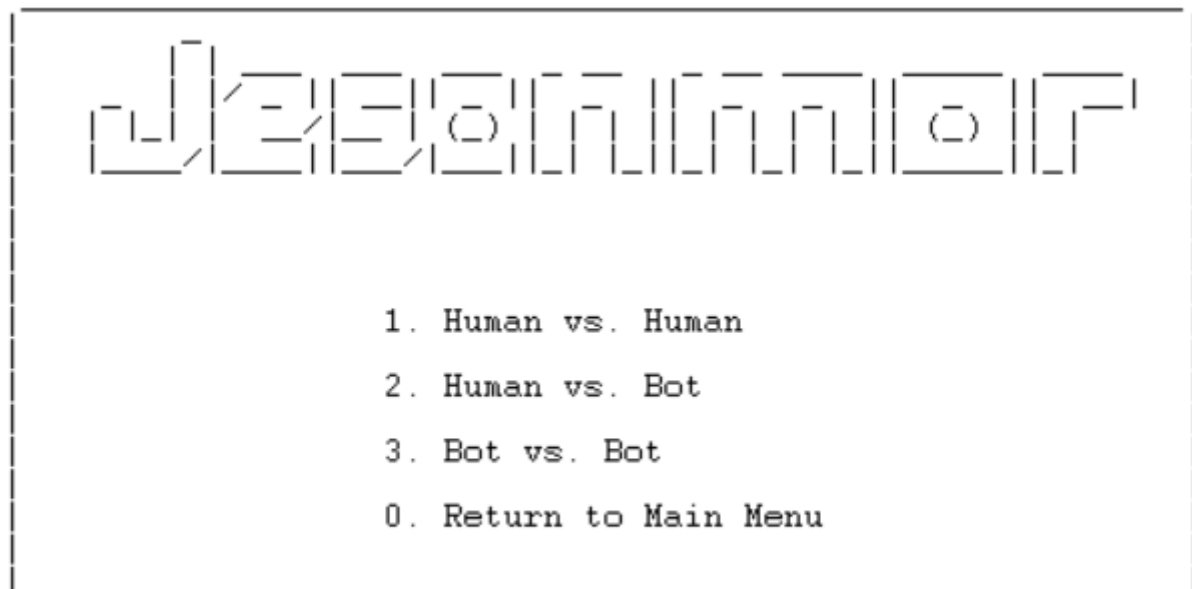
Importante: Para fazer a escolha de uma opção o utilizador deverá escrever o número relativo à opção que quer e clicar Enter.

- Logo a seguir a ter feito **play**, irá aparecer o menu principal do jogo:



Insert option:
-> ■

- A opção **1** mostra o seguinte menu que contém as 3 seguintes opções



Insert option:
=>

Neste Menu se selecionar a opção 2 ainda terá de escolher quem joga primeiro no seguinte menu

(caso escolha a opção 1 ou 3 o jogo será iniciado logo de seguida)




Who goes First?

1. Human (O)
2. Bot (X)
0. Return Back

Insert option:

->

- A opção 2 apenas contém um texto descritivo sobre as regras do jogo caso o utilizador queira saber.



Rules

HORSES - Each horse moves like the Chess knight (i.e., one cell on an orthogonal direction then one cell diagonally forward on that direction; it may jump).

GOAL - Wins the player that first occupy the center cell and then leaves it or captures all enemy horses.

0. Go Back

Insert option:

->

- A opção 3 contém uma breve mensagem de despedida e em 5 segundos limpa o ecrã e sai do jogo



Jogo

Assim que inicie o jogo irá aparecer o tabuleiro:

	1	2	3	4	5	6	7	8	9
A	O	O	O	O	O	O	O	O	O
B									
C									
D									
E					?				
F									
G									
H									
I	X	X	X	X	X	X	X	X	X

E, dependendo se é a vez do jogador ou do bot, apresenta um diálogo a pedir input ou um diálogo com o jogada que o bot irá fazer.

4.3 Execução de Jogadas:

Para o utilizador conseguir executar uma jogada, o programa tem de retornar corretamente de dois predicados:

- `choose_move(GameState, Size, Player, 'Player', [SelectedPosition, MovePosition])`

- `move(+GameState, +Player, +Move, -NewGameState)`

No primeiro destes 2 predicados é selecionado para onde o jogador quer mexer e esses valores são utilizados no `move`. O `move` recebe as duas posições [`SelectedPosition`, `MovePosition`] que é passado como `Move`, a selecionada e a posição para onde se vai mover. A seguir o predicado `getSelAndMovePosition` divide esse `move` em duas posições a inicial e a final. Depois substituímos na posição inicial com um espaço que representa um espaço vazio e na posição final substituímos com a peça que queremos mexer.

As jogadas do bot são feitas de maneira semelhante. O programa cria primeiro a lista com as jogadas possíveis para as peças do bot, selecionando aleatoriamente uma jogada dessa lista, movendo a peça de acordo com a jogada escolhida.

4.4 Final do Jogo:

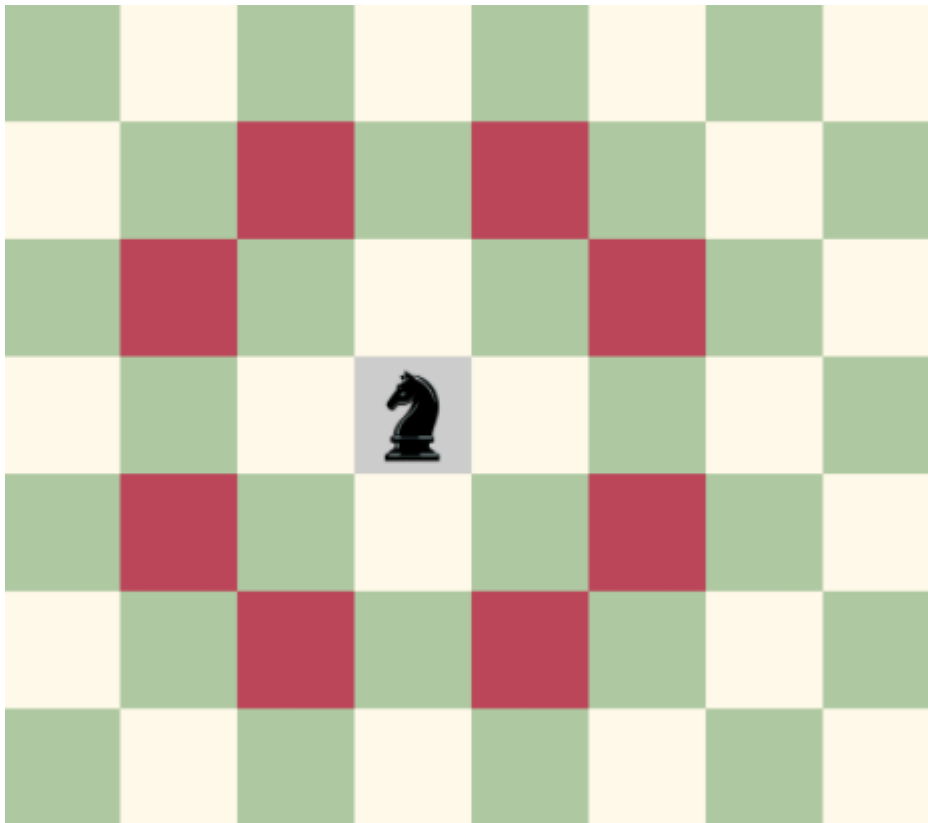
Um caso possível de acontecer é de um utilizador completar a sua jogada para o meio, mas ser comido na jogada a seguir e, caso seja a última peça, perder o jogo. Para verificar esta ocorrência, no final de uma jogada, o predicado `game_over(+GameState, +Player, -Winner)` é chamado e primeiro verifica se há peças do jogador ainda no tabuleiro e a seguir se estava a peça mexida previamente na posição central do tabuleiro. Caso alguma das 2 situações se confirmarem, o jogo acaba.

```
%Para verificação de existencia de peças
checkWinner(GameState, Size, Player, Winner) :-
    getPlayerPieces(GameState, Size, Player, ListOfPositions),
    isEmpty(ListOfPositions),
    Winner is -Player.

%Para verificar se a peça mexida estava previamente no centro
checkWinner(_, _, Player, Move, Winner):-
    getSelAndMovePosition(Move, SelRow-SelColumn, _),
    SelRow is 4,
    SelColumn is 4,
    Winner is Player.
```

4.5 Lista de Jogadas Válidas:

- Uma jogada é composta por duas posições no tabuleiro. Ambas são compostas por um valor correspondente a uma coluna (inteiro) e um valor correspondente a uma linha (caráter). A primeira será a posição da peça que se quer mexer e a segunda para onde a quer mover.



Posições a vermelho são os movimentos legais considerando que não sejam ou peças do mesmo jogador ou fora de limites

- Depois de ambos os inputs serem feitos, o valor da linha em ambos é transformado num Inteiro correspondente ao índice da linha começando em 0, para facilitar os calculo no back-end.
- O predicado `valid_moves` usa o predicado `getPlayerPieces` que verifica recursivamente, posição a posição, começando na posição (0,0) até a posição (8,8) que posições têm uma peça do jogador e guarda as numa lista. Se a posição que o jogador inseriu primeiro se encontra em tal lista então confirmamos que o primeiro input é válido. Logo a seguir chama o predicado `getPossibleMoves` que retorna, por sua vez utilizando o mesmo sistema de recursão, uma lista de todas as posições para qual legalmente a peça selecionada se pode mexer e verifica se uma delas é a posição que o jogador inseriu em segundo lugar. Para além destes predicados, temos o predicado auxiliar `checkMove` que serve para verificar quais das 8 posições a volta da posição selecionada esta disponivel. É usada dentro do predicado `getPossibleMoves`.

```
choose_move(GameState, Size, Player, 'Player', [SelectedPosition, MovePosition]):-
    valid_moves(GameState, Size, Player, _),
    selectPiece(GameState, Size, Player, SelectedPosition),
    movePiece(GameState, Size, Player, SelectedPosition, MovePosition).

valid_moves(GameState, Size, Player, ListOfMoves):-
    getPlayerPieces(GameState, Size, Player, ListOfPositions),
    getPossibleMoves(GameState, Size, Player, ListOfPositions, ListOfMoves),
    \isEmpty(ListOfMoves).
```

5. Conclusões

Limitações do trabalho e RoadMap

- Nas fases mais iniciais do desenvolvimento do jogo encontramos nos com vários problemas relacionados a "tradução" do nosso raciocínio para a linguagem prolog vista que é uma linguagem bastante diferente de outras que estávamos mais habituados mas a seguir dessa fase inicial conseguimos desenvolver um trabalho do qual nos sentimos orgulhosos.
- As seguintes fase de desenvolvimento seria criar o nível 2 do bot e aprofundar nos diferentes níveis de "inteligência" que o bot poderia ter, como por exemplo, guardar em memória os movimentos mais comuns do adversário em certas situações e analisar duas jogadas à frente. Também deveríamos ter um sistema de avaliação do estado de jogo em cada instante para ajudar o jogador.

6. Bibliografia

Todas as referencias abaixo foram meramente utilizadas para dar nos conhecimento sobre o jogo e/ou linguagem prolog

- Sterling, L. and Shapiro, E., 1986. The Art of Prolog. 2nd ed.
- <http://www.di.fc.ul.pt/~jpn/gv/jesonmor.html>
- <https://github.com/Forensor/jeson>
- Slides de aulas teóricas