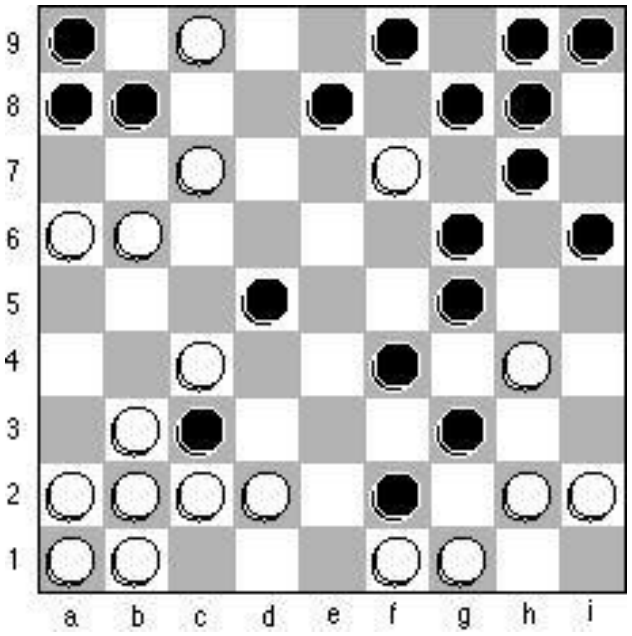
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Trabalho Prático 2

Renpaarden



Programação Funcional e em Lógica 2021/2022

Grupo: Renpaarden_2

Filipe Campos up201905609@edu.fe.up.pt
Francisco Cerqueira up201905337@edu.fe.up.pt

Índice

Instalação e Execução

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

Descrição do Jogo

Renpaarden é um jogo de tabuleiro 9x9, em que cada jogador começa com 18 peças, 9 em cada um das linhas traseiras.
Um jogador deverá mover uma das suas peças, tal como um cavalo de Xadrez, tal como ilustrado na figura:

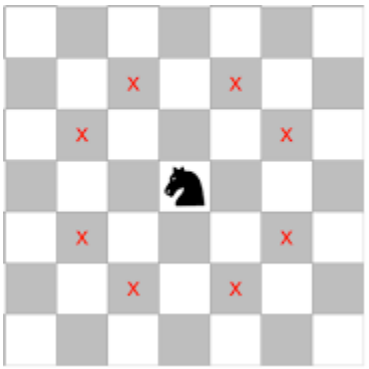


Figura 1 - Movimento da peça

Uma peça pode ser movida para uma posição vazia ou para uma posição ocupada pelo adversário, sendo neste caso necessário realizar outro movimento até atingir uma posição vazia.
O primeiro jogador que colocar todas as suas peças na posição inicial das do oponente, vence o jogo.

Lógica de Jogo

Representação Interna do Estado de Jogo

Tabuleiro

O tabuleiro é representado usando uma lista de sublistas, sendo que cada sublista representa uma linha horizontal do mesmo. Os valores das sublistas podem assumir 3 valores:

- **0** - Posição Vazia.
- **1** - Posição com peça pertencente ao jogador 1.
- **2** - Posição com peça pertencente ao jogador 2.

Por exemplo, o estado inicial do tabuleiro será representado da seguinte forma:

Inicial	Intermédio	Final

`  a b c d e f g h i`
`-------------------`
`1 | 2 2 2 2 2 2 2 2 2`
`2 | 2 2 2 2 2 2 2 2 2`
`3 | 0 0 0 0 0 0 0 0 0`
`4 | 0 0 0 0 0 0 0 0 0`
`5 | 0 0 0 0 0 0 0 0 0`
`6 | 0 0 0 0 0 0 0 0 0`
`7 | 0 0 0 0 0 0 0 0 0`
`8 | 1 1 1 1 1 1 1 1 1`
`9 | 1 1 1 1 1 1 1 1 1`
	`  a b c d e f g h i`
	`-------------------`
	`1 | 0 1 0 2 0 1 0 0 2`
	`2 | 0 0 0 0 0 2 1 1 1`
	`3 | 2 1 2 2 1 0 0 0 1`
	`4 | 1 0 1 2 0 0 0 0 0`
	`5 | 0 2 0 0 0 2 1 2 1`
	`6 | 0 0 0 1 1 1 2 1 0`
	`7 | 2 0 2 0 1 0 0 0 2`
	`8 | 0 0 2 2 0 2 0 0 0`
	`9 | 1 0 0 0 0 0 0 0 2`
		`  a b c d e f g h i`
		`-------------------`
		`1 | 1 1 0 1 1 1 0 1 1`
		`2 | 1 0 1 1 1 0 1 1 1`
		`3 | 0 0 0 0 0 0 0 0 0`
		`4 | 0 0 0 0 0 0 1 0 0`
		`5 | 0 0 1 0 0 0 0 0 0`
		`6 | 0 0 0 0 1 0 1 0 0`
		`7 | 0 0 0 0 0 0 0 0 0`
		`8 | 2 2 2 2 2 2 2 2 2`
		`9 | 2 2 2 2 2 2 2 2 2`

Tipo de Jogador

O tipo de jogador é identificado através do valor tomado pelo átomo PlayerType:

- `human`: Indica que o jogador é humano e qualquer ação envolvente no jogo requer input por parte do mesmo.
- `computer-Level`: Indica que o jogador é controlado pelo programa, realizando jogadas de acordo com o seu Level:
  - `1`: Das jogadas válidas disponíveis, escolhe uma destas aleatoriamente.
  - `2`: Através de um algoritmo mais robusto que avalia o estado do jogo, escolhe a que garante uma melhor jogada.
  - `3`: Semelhante ao anterior, mas tem em conta jogadas mais complexas (repetir a jogada, caso a peça calhe numa posição preenchida por uma peça adversária).

Estado de Jogo

O nosso projeto agrega, assim, o tabuleiro e o jogador atual numa só estrutura `game_state(Board, Player)`.

Jogabilidade

O nosso projeto disponibiliza 4 modos de jogo:

- **Humano** vs **Humano**
- **Humano** vs **Computador**
- **Computador** vs **Humano**
- **Computador** vs **Computador**

Nos casos em que o modo do jogo envolve pelo menos um oponente controlado pelo computador, são disponibilizados **2** níveis de dificuldades:

- **Fácil - Nível 1**
- **Intermédio - Nível 2**
- **Difícil - Nível 3**

Visualização do Estado de Jogo

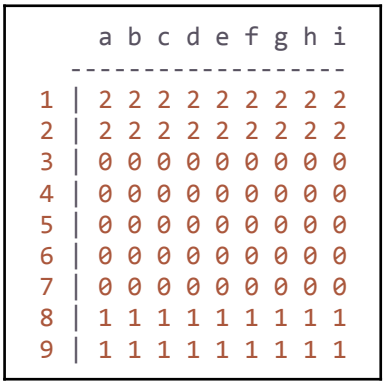
Após iniciar o jogo com o predicado `play.` o jogador tem ao seu dispor um menu inicial com as opções principais do jogo:

Figura 2 - Menu do jogo

O jogador apenas necessita de escrever o número relativo à opção que desejar e premir `Enter`.

Após selecionar qualquer uma destas (exceto 0), o jogo inicia, sendo a vez do jogador 1 a jogar.
No caso de pelo menos um dos jogadores se tratar de um computador, é necessário escolher a dificuldade do computador.

Assim que um jogo é iniciado, o tabuleiro é apresentado:



Sucedida a apresentação do tabuleiro, dependendo se é a vez do jogador ou do computador, é apresentado um diálogo a pedir input ou um diálogo com o jogada que o computador efetuará.

Execução de Jogadas

Para validar um move utilizamos a função `valid_move(+GameState, ?Move)`, que verifica se a posição de origem contém uma peça do jogador e se a peça escolhida pode ser movida para o destino pretendido através do uso da função `valid_position_for_piece(+GameState, +Piece, +Destination)`.

Esta, por sua vez, utiliza duas regras: `valid_position_for_piece_without_jump(+GameState, +Piece, +Destination)`, que confirma se o Destino é uma posição 'vizinha' da peça (Figura 1), e `valid_position_for_piece_with_jump(+GameState, +Piece, +Destination, +Visited)`, que aplica a regra de ser possível 'saltar' em cima de peças do adversário e realizar outro movimento.
Esta separação é necessária, pois os dois primeiros níveis de inteligência não são capazes de utilizar esta segunda regra.

Após realizada a validação de uma jogada (`Move`), basta utilizar `move(+GameState, +Move, -NewGameState)`, que retorna um novo GameState com os valores da board alterados, recorrendo ao predicado `change_value(+Board, +Position, +NewValue, -NewBoard)`, e com o turno do próximo jogador.

Final do Jogo

Como referido nas regras, o jogo termina quando um dos jogadores conseguir colocar todas as suas peças na posição original das peças do adversário.
Para realizar esta verificação utilizamos o predicado `game_over(+GameState, -Winner)`, que usufrui da funcionalidade de Pattern Matching de Prolog para realizar esta verificação de forma eficiente.

```
    a b c d e f g h i          a b c d e f g h i
  -------------------        -------------------
1 | 1 1 1 1 1 1 1 1 1      1 | _ _ _ _ _ _ _ _ _
2 | 1 1 1 1 1 1 1 1 1      2 | _ _ _ _ _ _ _ _ _
3 | _ _ _ _ _ _ _ _ _      3 | _ _ _ _ _ _ _ _ _
4 | _ _ _ _ _ _ _ _ _      4 | _ _ _ _ _ _ _ _ _
5 | _ _ _ _ _ _ _ _ _      5 | _ _ _ _ _ _ _ _ _
6 | _ _ _ _ _ _ _ _ _      6 | _ _ _ _ _ _ _ _ _
7 | _ _ _ _ _ _ _ _ _      7 | _ _ _ _ _ _ _ _ _
8 | _ _ _ _ _ _ _ _ _      8 | 2 2 2 2 2 2 2 2 2
9 | _ _ _ _ _ _ _ _ _      9 | 2 2 2 2 2 2 2 2 2
```

Lista de Jogadas Válidas

Para obter a lista de todas as jogadas válidas, utilizamos o predicado `valid_moves(+GameState, -Moves)`, e, para obter todas as jogadas válidas sem o uso da regra de saltar em peças do adversário, recorremos ao predicado `valid_moves_without_jump(+GameState, -Moves)` (AI nível 1 e 2).

Ambos estes predicados recorrem ao uso de `findall`, juntamente com os predicados de validação acima apresentados.

Avaliação do Estado do Jogo

O valor do estado de jogo, calculado através de `value(+GameState, -Value)`, é a soma dos valores individuais de cada peça do jogador que realizou o Move anterior. O valor de cada peça, obtido por `piece_value(+Piece,+Player,-Value)`, corresponde à distância da peça ao lado mais distante do adversário no tabuleiro. Portanto, quanto menor o valor do `GameState`, mais próximo este estará de uma vitória.

Jogada do Computador

- **Algoritmo AI nível 1** → Seleção aleatória de um dos movimentos válidos.
- **Algoritmo AI nível 2** → É escolhido aleatoriamente uma jogada a partir das jogadas que partilham o valor mínimo.
- **Algoritmo AI nível 3** → Utiliza o mesmo algoritmo que o nível anterior, com a diferença de que recorre a uma lista de jogadas mais complexa, que envolve os saltos descritos anteriormente.

Conclusões

No início do trabalho, sentimos dificuldades em estruturar os diferentes módulos necessários para implementar o jogo e a representação pretendida das peças.

Quanto a possíveis melhorias que faríamos ao jogo, caso tivéssemos mais tempo e fôssemos valorizados por tal, seria a nível estético, tanto da interface do menu como do jogo, visto que, por exemplo, poderiam ser utilizadas cores e designs diferentes para que o utilizador tivesse uma melhor experiência. No entanto estas alterações seriam limitadas pelo facto de que o programa é executado através de uma consola.

Bibliografia

A. https://boardgamegeek.com/boardgame/70925/renpaarden
B. https://www.di.fc.ul.pt/~jpn/gv/renpaarden.htm
C. https://sites.google.com/site/boardandpieces/list-of-games/renpaarden