Trabalho de Programação Lógica 2021

Trabalho Five Field Kono - Grupo 3

- André Diogo Bastos Pereira - up201905650 (50%)
- Matilde Jacinto Oliveira - up201906954 (50%)

Instalação e Execução

De forma a executar o jogo é apenas necessário fazer a consulta do ficheiro `game.pl` no sicstus. E chamar o predicado `play/0`.

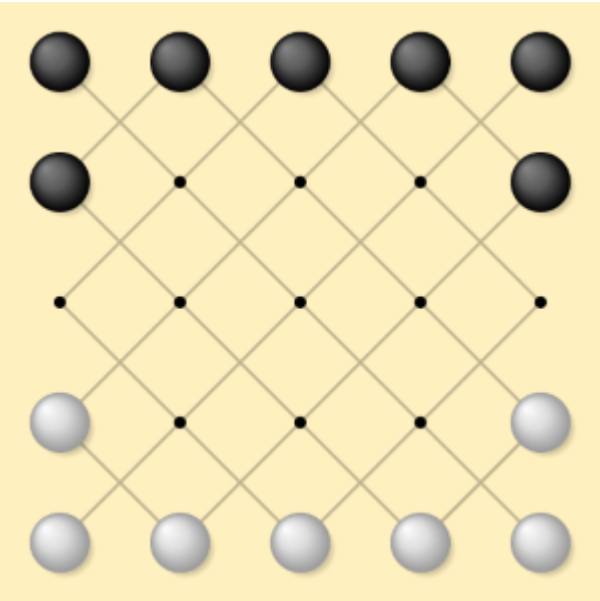
Descrição do jogo

Five Field Kono é um jogo de estratégia abstrata coreano, para dois jogadores, em que o objetivo é que o ganhe o jogador que conseguiu mover todas as suas peças desde os locais de inicias até aos locais inicias das peças do adversário.

De forma muito simplista, a lógica do jogo consiste em as jogadas sejam alternadas entre os dois jogadores, sendo que numa jogada o jogador é obrigado a mover uma das suas peças para uma posição que esteja na diagonal da atual e à distância de uma casa, havendo no máximo 4 jogadas possíveis, podendo andar para trás.

O jogo acaba mal um dos jogadores consiga que as suas peças fiquem todas nas posições das peças inicias do adversário.

Exemplo de um board é apresentado na fotografia em baixo.



Lógica de Jogo

Representação interna do estado de jogo

- **'B'**: Black (peças pretas)

- **'W'**: White (peças brancas)
- **'E'**: Empty (lugares vazios)
- **Board**:

exemplo de estado inicial do tabuleiro com tamanho 5.

```
['W',
['B','B','B','B','B'],
['B','E','E','E','B'],
['E','E','E','E','E'],
['W','E','E','E','W'],
['W','W','W','W','W']]
```

exemplo de estado intermédio do tabuleiro com tamanho 5.

```
['B',
['B','E','E','B','B'],
['B','B','E','E','B'],
['E','B','E','W','E'],
['W','E','E','W','E'],
['W','W','W','W','E']]
```

exemplo de estado final do tabuleiro com tamanho 5, onde ganha o jogador White.

```
['B',
['W','W','W','W','W'],
['W','E','E','E','W'],
['E','B','E','E','E'],
['E','E','E','B','B'],
['B','B','E','B','B']]
```

Primeiro elemento da lista representa o próximo jogador a jogar, no primeiro exemplo o jogador White ('W'). Note-se que começa sempre o jogador White a jogar, daí o estado inicial ter sempre o elemento 'W' como primeiro elemento da lista.

As posições das peças no Board, são representadas por duas coordenadas X e Y, que representam, respetivamente, o número da coluna (X) e o número da linha (Y), ambos iniciam em 0 e terminam no tamanho do Board menos um.

- **Move**: [[**x**,**y**], [**deltaX**, **deltaY**]]
  - **X**: 0 <= X < Tamanho do Board
  - **Y**: 0 <= Y < Tamanho do Board

- **deltaX**: 1 ou -1
- **deltaY**: 1 ou -1

Exemplo: [[0,0] , [1,1]]

As moves são representadas por a posição atual da peça que se quer mover e por um delta, que indica quanto é que a peça se desloca em X e em Y (sendo que apenas pode tomar como valor: 1 e -1, pois apenas pode andar na diagonal e uma casa).

Visualização do estado de jogo

De forma a tornar o menu fácil de utilizar procuramos tornar a interação com o utilizador bastante simples. Para tal, procuramos reduzir ao máximo os caracteres que o utilizador tem de escrever. Obtemos esse resultado através sistemas de menus como é possivel ver no menu principal do jogo em baixo:

```
White Player: human          Black Player: computer-2
Board size: 5

MENU
1. Start Game
2. Change White Player
3. Change Black Player
4. Change Board Size
5. Quit

Option:
```

De forma a modificar os jogadores associados ás peças pretas ou brancas, é necessário escolher a opção para tal, e realizar a escolha desejada nos menus que se sucedem `input_player/1` e `set_player/2`. Para validar o input do utilizador é sempre verificado se o número dado está na gama de valores desejada. Caso este não seja válido, a pergunta será apresentada novamente. Adicionalmento no começo da aplicação, os jogadores default são humano contra humano.

Quando é pretendido modificar o tamanho do tabuleiro a gama de valores é também verificada, mas o input é lido com o predicado `read_number/2` desenvolvido em aula já que este pode apresentar mais do que um dígito.

Quanto ao desenrolar do jogo, para manter o utilizador atualizado do estado de jogo o tabuleiro é escrito no ecrã após cada jogada, juntamente com o proximo jogador `WHITE/BLACK` a jogar. Sendo o seu estilo o seguinte:

```
**          Five Field Kono          **

      A   B   C   D   E
    +---+---+---+---+---+
  a | B | B |   | B | B |
    +---+---+---+---+---+
  b |   |   | W | B | B |
    +---+---+---+---+---+
  c |   | B |   |   |   |
    +---+---+---+---+---+
  d |   | W | W |   | W |
    +---+---+---+---+---+
  e |   |   | W | W | W |
    +---+---+---+---+---+

Next Player: WHITE

Want to see which are your valid moves (y)? n

PIECE TO MOVE (a-A): d-C

WHERE TO MOVE THE CHOSEN PIECE (a-A): 
```

O tabuleiro poderá variar no tamanho, como foi mostrado nas opções do menu em cima e este ficará como mostrado no exemplo em baixo:

```
**          Five Field Kono          **

      A   B   C   D   E   F   G   H   I   J
    +---+---+---+---+---+---+---+---+---+---+
  a | B | B | B | B | B | B | B | B | B | B |
    +---+---+---+---+---+---+---+---+---+---+
  b | B |   |   |   |   |   |   |   |   | B |
    +---+---+---+---+---+---+---+---+---+---+
  c |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  d |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  e |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  f |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  g |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  h |   |   |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+---+---+
  i | W |   |   |   |   |   |   |   |   | W |
    +---+---+---+---+---+---+---+---+---+---+
  j | W | W | W | W | W | W | W | W | W | W |
    +---+---+---+---+---+---+---+---+---+---+

Next Player: WHITE
```

Quanto à escolha de uma jogada, o input deverá ser dado em letras, sendo que, para tal será necessário fornecer ao predicado `ask_piece/2` um input como o que está representado no exemplo `a-A`. Uma `letra minúscula`, que representa a linha da peça escolhida, seguida do separador `-`, e uma `LETRA MAIÚSCULA`, que representa a coluna da peça. Caso a posição escolhida contenha uma peça válida o predicado `ask_move/2` será chamado e o utilizador terá de indicar qual o buraco para onde deseja mover a peça seguindo o mesmo estilo.

Adicionalmento o utilizador poderá vizualizar todas as jogadas válidas no momento. Para tal terá que responder com o character `y` à primeira pergunta de cada novo pedido de jogada (em `ask_move_show/2`), sendo estas listadas da seguinte forma:

```
**          Five Field Kono          **

        A   B   C   D   E
      +---+---+---+---+---+
    a |   |   |   | B | B |
      +---+---+---+---+---+
    b | B | B | B | B | B |
      +---+---+---+---+---+
    c |   | W |   | W |   |
      +---+---+---+---+---+
    d |   |   | W |   | W |
      +---+---+---+---+---+
    e | W |   | W |   | W |
      +---+---+---+---+---+

Next Player: BLACK

Want to see which are your valid moves (y)? y

Piece on: b-A to a-B
Piece on: b-B to a-A
Piece on: b-B to a-C
Piece on: b-B to c-A
Piece on: b-B to c-C
Piece on: b-C to a-B
Piece on: b-D to a-C
Piece on: b-D to c-C
Piece on: b-D to c-E
```

Execução de jogadas

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

A execução de uma jogada consiste em essencialmente dois passos:

1. Validação da jogada (`validate_move/2`), verificando se as coordenadas dadas correspondem a uma peça do jogador que é suposto jogar, em `validate_player/2`, e se a translação, definida no segundo argumento da move, é de apenas uma casa (verificado com o predicado `delta/2`) e leva a peça para um lugar que está vazio ('E'), em `validate_shift/2`.
2. Execução da jogada, recorrendo ao predicado `make_move/3`. Este calcula, substitui a posição atual da peça a jogar por um 'E' e substitui a posição para o qual a peça vai pelo identificador do jogador que jogou. Estas substituições realizam-se através do predicado `replace_element/4`, que na lista, troca o element numa certa posição por outro.

No término da execução, é alterado o jogador a jogar.

Exemplo:

```
move(['B',
['B','B','B','B','B'],
['B','E','E','E','B'],
['E','W','E','E','E'],
['E','E','E','E','W'],
['W','W','W','W','W']],
[[0,0],[1,1]],
['W',
['E','B','B','B','B'],
['B','B','E','E','B'],
```

```
      ['E','W','E','E','E'],
      ['E','E','E','E','W'],
      ['W','W','W','W','W']])`
```

Final do jogo

`game_over(+GameState, -Winner)`

Este predicado é chamado no início de cada ciclo de jogo, prevendo se o jogo continua ou se a jogada anterior fez com que um dos jogadores atingisse o objetivo do jogo.

Assim, se for o jogador **white ('W')** a jogar, este predicado vai verificar se a move anterior fez com que o jogador **black ('B')** enchesse todos os seus buracos finais e terminasse o jogo.

Esta verificação é feita percorrendo a última linha do board e os dois espaços, inicial e final, da penúltima linha. A validação da presença de uma peça numa posição de uma lista é feita pelo predicado `pos_element/3`.

Se assim for, o winner será o 'B', no predicado em causa (`game_over/2`).

O inverso é realizado se for a vez do jogador **black ('B')** jogar, indicando que o **white ('W')** acabou de o fazer e pode ter terminado o jogo.

Lista de jogadas válidas

`valid_moves(+GameState, -ListOfMoves)`

O predicado valid_moves que comporta todas as jogadas válidas (do estilo representado na secção - Representação interna do estado de jogo), foi implementado seguindo a seguinte estrutura:

1. Mediante o jogador a jogar, representado no primeiro elemento de GameState, o predicado `player_pieces(+Player, +Board, -Positions)` irá coletar todas as posições atuais das peças desse jogador.
2. Depois o predicado `go_through_moves(+GameState, +Pieces, -ValidMoves)` faz uma pesquisa com recurso ao predicado aprendido `findall/3` que identificará quais destas jogadas são válidas, com recurso ao já apresentado predicado `validate_move/2`.

A utilização do predicado em estudo, que recolhe todas as jogadas válidas para um certo jogador no momento, depende do tipo de jogador em causa:

- No caso do jogador a jogar ser do tipo `human`, no momento da sua jogada, antes de receber inputs relativamente à execução de uma jogada, é questionado se gostaria de ver as suas jogadas possíveis. No caso de resposta afirmativa no predicado `show_moves('y', +GameState)` é chamado o `valid_moves/2` com posterior display das jogadas em `show_moves/1`.
- No caso de um jogador do tipo `computer`, a questão não se coloca e o `show_moves/2` não calcula as jogadas válidas nesse momento, mas sim no momento de escolha de uma jogada para o bot.

Avaliação do estado de jogo *

`value(+GameState, -Value)`

O algoritmo de avaliação do estado de jogo atual foi desenvolvido através de uma técnica de algoritmo míope que escolhe a jogada seguinte com base na escolha atual ótima, na esperança de encontrar um ótimo global que o leve á vitória.

Nesse sentido, a estratégia utilizada para a avaliação foi:

- Descobrir quais as posições no tabuleiro, onde o jogador tem que estar para o jogo acabar, que estão até ao momento vazias (com `get_empty_final/3`);
- Descobrir quais as peças desse jogador que ainda não estão em lugares finais (com `get_filled_board/3`);
- Para cada posição final vazia descobrir qual a peça que está mais perto e que consegue chegar a esse buraco (visto que como as peças apenas se deslocam na diagonal não consegue chegar a todos os buracos do tabuleiro*), no predicado `small_distance/4`;
- Após o matching entre peça e buraco final é calculado para os pares a distância^2 entre os dois (D = (X1+X2)^2 + (Y1+Y2)^2).
- O **valor final do tabuleiro** é calculado com a **soma de todas as distâncias mais um incremento de 100 por cada peça que ainda não está numa posição final**, levando a que quanto mais peças maior o valor e pior a avaliação do board, valorizando a presença de peças nos buracos finais.

> Nota: verificação da possibilidade de uma peça conseguir atingir uma posição final, feita nesta condição: (((X mod 2) + (Y mod 2)) mod 2) =:= (((X1 mod 2) + (Y1 mod 2)) mod 2). Que basicamente cumpre com: uma casa com ambas as coordenadas pares ou ambas ímpares só consegue chegar a uma casa com coordenadas também ambas pares ou ambas ímpares. No caso de uma coordenada ser ímpar e outra par, estas peças só atingem peças em posições cujas coordenadas têm também um valor ímpar e outro par.

Jogada do Computador

`choose_move(+Level, +GameState, +Moves, -Move)`

Após recolhidas as jogadas válidas para o jogador atual é utilizada a função `choose_move\4`. Esta função terá diferentes abordagens dependendo do `Level` que receber, 1 ou 2.

- No caso de `Level` 1 será escolhido, com o predicado `random_select\3`, uma `Move` aleatória de todas as `Moves` fornecidas e válidas.

- No caso de `Level` 2 será calculado o `Value` associado ao board após a realização de cada jogada da lista de `Moves`. Estes valores à medida que vão sendo calculados serão ordenados devido ao uso do predicado `setof`. Com a lista de moves organizada, fazemos a chamada ao predicado `randomize_best_moves/2` que escolhe uma `Move` aleatória entre todas as `Moves` com o menor value existente.

> Nota: ao jogar computador vs computador há uma pequena possibilidade que estes se bloqueiem um ao outro a partir de um certo momento do jogo, visto que a jogada ótima é única e bloqueia o adversário, obrigando o a recuar.

Conclusões

Concluímos que este trabalho foi importante para adquirirmos os principais conhecimentos da linguagem lógica de prolog e permitiu-nos dar os primeiros passos num paradigma que consideramos útil e importante

para o nosso desenvolvimento como engenheiros informáticos.

Já que o desenvolvimento do trabalho coincidiu com a aprendizagem contínua da linguagem, houveram decisões que foram tomadas no início do desenvolvimento que, se iniciássemos agora o trabalho, não teríamos implementado as coisas dessa forma. Um exemplo claro desta situação é a nossa representação do estado de jogo, com cada elemento do Board estar representado em letra maiúscula dentro de plicas ou o player ser o primeiro elemento da lista GameState em vez do GameState ser por exemplo Player-Board. Como só tomamos conhecimento desta situação mais perto do final, não fez sentido alterar já que funcionava de igual forma eficiente.

Como é sabido, neste momento estamos a realizar outros projetos e se não fosse esta sobreposição de tarefas, gostaríamos de ter melhorado ou implementado outros níveis de dificuldade para um jogo contra o computador.

No desenrolar da implementação do jogo, houveram duas questões que consideramos ser menos intuitivas que gostávamos de destacar. Em primeiro lugar, o trabalho com input de informação é diferente e por vezes foi difícil perceber como obter o resultado que esperávamos. Em segundo, a tomada de decisão de como iríamos proceder à avaliação do Board foi difícil, dado que no jogo em questão não havia pontuação envolvida e por vezes o cálculo de distâncias poderia levar a ciclos indesejados.

Bibliografia

https://boardgamegeek.com/boardgame/25471/five-field-kono

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

https://www.swi-prolog.org/

https://sicstus.sics.se/sicstus/docs/latest/html/sicstus/The-Prolog-Library.html