U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

2019-2020

Relatório de avaliação intercalar

Programação em Lógica – Grupo 2

quantik

João Henrique Afonso Marques Reguengo da Luz - 201703782

Liliana Natacha Nogueira Almeida - 201706908

Índice

Introdução

Este trabalho teve como objetivo o desenvolvimento, em linguagem *Prolog*, de um jogo de tabuleiro, para dois jogadores, denominado por *Quantik*. É caracterizado pelo tipo de tabuleiro e de peças, pelas regras de movimentação das peças (jogadas possíveis) e pelas condições de terminação do jogo com derrota, vitória ou empate.

É possível jogar *Quantik* em três modos diferentes: Humano/Humano, Humano/Computador e Computador/Computador, com dois níveis de dificuldade para o computador.

O relatório apresenta a seguinte estrutura:

- **O Jogo *Quantik*:** Descrição sucinta do jogo, da sua história e das suas regras.
- **Lógica do Jogo:** Descrição do projeto e da implementação da lógica do jogo.
    - **Representação do Estado do Jogo:** Exemplificação dos estados iniciais, intermédios e finais do jogo.
    - **Visualização do Tabuleiro:** Descrição do predicado de visualização do tabuleiro.
    - **Lista de Jogada Válidas:** Obtenção de uma lista de jogadas possíveis.
    - **Execução de Jogadas:** Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo.
    - **Final do Jogo:** Verificação do fim do jogo, com identificação do vencedor.
    - **Avaliação do Tabuleiro:** Forma(s) de avaliação do estado do jogo.
    - **Jogada do Computador:** Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade.

O Jogo *Quantik*

Quantik é um jogo de tabuleiro puramente estratégico lançado em agosto de 2019. O objetivo de cada jogador é ser o primeiro a colocar uma peça que forme uma linha, coluna ou quadrante constituído pelas quatro peças distintas.

O jogo é constituído por um tabuleiro quadrado de dimensões 4x4. Por cada jogador são distribuídas duas peças de cada sólido geométrico diferente: 2 cilindros, 2 cubos, 2 cones e 2 esferas. Em cada jogada, os jogadores colocam alternadamente uma das suas peças no tabuleiro. Não é permitido colocar uma peça numa linha, coluna ou quadrante no qual já exista uma peça do adversário com a mesma forma. No entanto, a peça pode ser colocada se ambas pertencerem ao mesmo jogador.

O jogo termina quando um dos jogadores colocar a quarta peça distinta numa linha, coluna ou quadrante, independentemente das peças pertencerem todas a esse jogador.

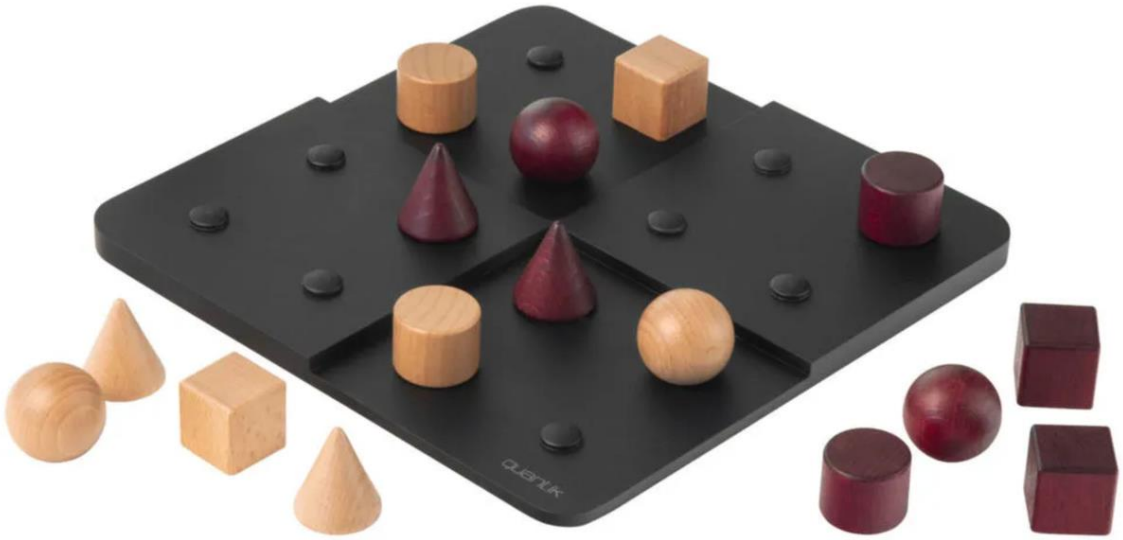

Figura 1 - Exemplo do tabuleiro de um jogo a decorrer.

Lógica do Jogo

Representação interna do estado do jogo

Cada tipo de peça tem uma nomenclatura única que indica de uma forma intuitiva o tipo de sólido e o jogador a que a peça pertence.

A correspondência entre os átomos e as peças é a seguinte:

wCyl → Cilindro branco bCyl → Cilindro preto
wCub → Cubo branco bCub → Cubo preto
wCon → Cone branco bCon → Cone preto
wSph → Esfera branca bSph → Esfera preta

Estado inicial:

```
initial_board([
    [empty, empty, empty, empty],
    [empty, empty, empty, empty],
    [empty, empty, empty, empty],
    [empty, empty, empty, empty]
]).
```

Figura 2 - Representação em Prolog do tabuleiro do jogo no estado inicial.

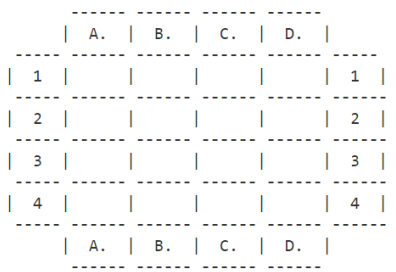

Figura 3 - Representação visual do estado inicial do tabuleiro.

Estado intermédio:

```
mid_board([
    [empty, cube_w, empty,  cil_b],
    [cil_w, sph_b,  empty,  empty],
    [empty, cone_b, cone_b, sph_w],
    [empty, empty,  cil_w,  empty]
]).
```

Figura 4 - Representação em Prolog do tabuleiro do jogo numa fase intermédia.

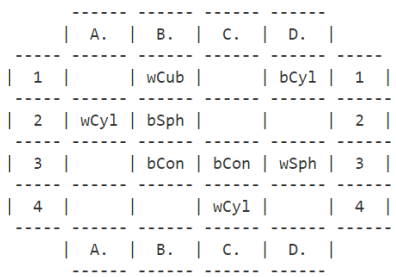

Figura 5 - Representação visual de um estado intermédio do tabuleiro.

Estado final:

```
final_board([
    [empty, cube_w, empty,  cil_b],
    [cil_w, sph_b,  empty,  empty],
    [empty, cone_b, cone_b, sph_w],
    [empty, empty,  cil_w,  cube_w]
]).
```

Figura 6 - Representação em Prolog do tabuleiro do jogo numa possível fase final.

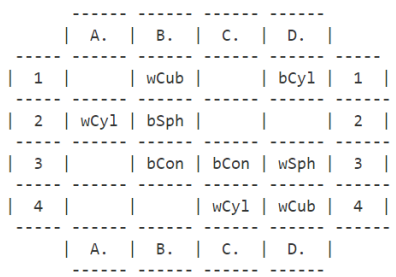

Figura 7 - Representação visual de um estado final do tabuleiro.

Visualização do tabuleiro

Para visualizar o tabuleiro em modo de texto é necessário utilizar o predicado **display_game** que recebe como argumentos o tabuleiro num dos seus estados e o próximo jogador a colocar uma peça. O output deste predicado pode ser visto nas figuras 3, 5 e 7.

```
display_game(Board, Player):-
    display_board(Board),
    nl,
    write('Next player: '),
    write(Player).
```

Figura 8 - Predicado de visualização do tabuleiro do jogo.

Lista de Jogadas Válidas

Em cada jogada é possível colocar uma peça no tabuleiro desde que sejam cumpridas regras estabelecidas, isto é, desde que não esteja a ser colocado um sólido na linha, coluna ou quadrante onde o adversário já tenha colocado peças do mesmo sólido. A escolha do sólido a colocar está ainda limitada pelo número de peças de cada tipo (duas por cada sólido, para cada jogador). Esta validação é feita pelo predicado **is_move_valid**.

A criação de uma lista composta por todas as jogadas válidas é feita pelo predicado **valid_moves** que gera, para cada posição, todas as jogadas possíveis para o atual jogador adicionando à lista aquelas que são consideradas válidas. As seguintes imagens mostram como é feito este processo:

```
valid_moves(Board, Player, AllBoards):-
    findall(NewBoard, random_move(Board, Player, NewBoard), AllBoards).
```

Figura 9 - Predicado que forma lista com todas as jogadas possíveis.

```
random_move(Board, Player, NewBoard):-
    generator_move(Row, Column, Piece, Player),
    is_move_valid(Row, Column, Piece, Board),
    move([Row, Column, Piece], Board, NewBoard).
```

Figura 10 - Predicado que gera uma jogada válida.

No predicado **random_move** é gerada uma jogada para uma determinada posição do tabuleiro tendo em conta o jogador atual. Se a jogada for válida, o tabuleiro é atualizado com a mesma.

Recorrendo ao predicado **findall**, é possível adicionar a uma lista (**AllBoards**), todos os tabuleiros que foram gerados e considerados válidos em **random_move.**

Execução de Jogadas

No predicado **game_loop**, encontra-se o ciclo principal do jogo responsável pela execução das jogadas e respetiva validação, assim como da verificação do estado de jogo.

A regra a consultar ao longo do jogo, depende do tipo de jogador atual, ou seja, são percorridas etapas diferentes conforme o jogador é humano ou não.

```
game_loop(Player, Board, h, NextPlayerType, Level):-
    repeat,

    % Move Human Piece
    once(human_move(Board,Player,NewBoard)),

    % Check Game Over
    (game_over(NewBoard, Player)
    ;
    next_player(Player, NewPlayer),
        % Checks for next turn valid moves
        (valid_moves(NewBoard, NewPlayer, AllBoards),

        % No more valid moves
        list_empty(AllBoards),
        display_tie(NewBoard)
        ;

        % Next player turn
        display_game(NewBoard, NewPlayer),
        game_loop(NewPlayer, NewBoard,  NextPlayerType, h, Level)
        )
    ).
```

Figura 11 - Predicado do ciclo principal do jogo para um jogador humano.

No caso de o jogador ser humano, em cada jogada, no predicado **human_move**, é-lhe pedido para especificar a peça a mover e a linha e coluna onde quer que esta seja colocada. Estas leituras de *input* são validadas em **validate_move_input**, confirmando se a peça pertence ao jogador e se as coordenadas fornecidas correspondem a uma posição válida do tabuleiro. O procedimento descrito apresenta-se na seguinte imagem:

```
validate_move_input(Row, Column, Piece, Player):-
    get_column(ColumnLetter),
    once(validate_column(ColumnLetter, Column)),
    get_row(Row),
    once(validate_row(Row)),
    get_solid(Solid),
    validate_solid(Solid, Piece, Player).
```

Figura 12 - Predicado que valida o input da jogada do utilizador.

Uma vez validada input dado pelo jogador, são feitas as devidas alterações ao tabuleiro recorrendo ao predicado **move**, no qual é verificado se a posição no tabuleiro está vazia, se não estão a ser colocados mais do que dois sólidos iguais e, por fim, se a jogada pretendida não viola nenhuma das regras previamente estabelecidas.

Por fim, é realizada a verificação do estado do jogo em **game_over**, referida na próxima secção. Se houver uma próxima jogada é dada a vez ao próximo jogador.

No caso de o jogador ser o computador, em vez do predicado **human_move** é usado o predicado **computer_move**, que procura todas as jogadas possíveis e escolhe uma, em função do nível de dificuldade escolhida.

Final do Jogo

No final de cada jogada, é necessário verificar se o jogo termina, isto é, se uma linha, coluna ou quadrante possui quatro sólidos distintos. Esta verificação do final do jogo é feita no predicado **check_game_over** demonstrado a seguir:

```
check_game_over(Board):-
    check_full_row(4, 4, Board)
    ;
    check_full_column(4, Board)
    ;
    check_full_quadrant(Board).
```

Figura 13 - Predicado que verifica o final do jogo.

Quando o final do jogo é alcançado, é apresentado o tabuleiro final e anunciado o vencedor se este existir. Para esse efeito, é utilizado o predicado **game_over**:

```
game_over(Board, Player):-
    check_game_over(Board),
    display_game_over(Board, Player).
```

Figura 14 - Predicado que coordena a verificação e representação do final do jogo.

Na eventualidade do jogador não concluir o jogo com a última jogada, pode ser declarado um empate no caso de não existirem jogadas possíveis para o próximo jogador.

Avaliação do tabuleiro

Para avaliar o estado do jogo foram tidas em conta todas as jogadas possíveis. Recorrendo ao predicado **evaluate**, todas estas jogadas são avaliadas e, no caso de resultarem na conclusão do jogo são adicionadas a uma lista.

```
evaluate_moves(AllBoards, WinningBoards) :-
    evaluate(AllBoards, _, WinningBoards).

evaluate([], WinningBoards, WinningBoards):- !.
evaluate([Board | Rest], WinningBoards, NewWinningBoards):-
    once(value(Board, Value)),
    once(add_board_by_value(Board, Value, WinningBoards, NewWinningBoards2)),
    evaluate(Rest, NewWinningBoards2, NewWinningBoards).
```

Figura 15 - Predicados que avaliam todas as jogadas e criam uma lista com as jogadas vencedoras.

A adição de cada jogada à lista de jogadas vencedoras é feita em função do valor atribuído pelo predicado **value**. No caso de a jogada concluir o jogo, o valor retornado é 1, caso contrário retorna 0.

```
value(Board, Value):-
    check_game_over(Board),
    good_move(Value)
    ;
    bad_move(Value).
```

Figura 16 - Predicado que atribui um valor a uma jogada.

Jogada do Computador

A seleção de jogadas por parte do computador é realizada no predicado **computer_move**.

```
computer_move(Board, Player, Level, FinalBoard):-
    valid_moves(Board, Player, PossibleBoards),
    choose_move(PossibleBoards, Level, FinalBoard).
```

Figura 17 - Predicado que despoleta a escolha da jogada do computador.

Este processo passa por reunir todas as jogadas possíveis, avaliá-las e escolher uma em função do nível de dificuldade escolhido pelo utilizador.

A avaliação das jogadas e consequente seleção, são desencadeadas pelo predicado **choose_move**, que utiliza para tal os predicados **evaluate_moves** (abordado na secção anterior) e **choose_one_board**, respetivamente.

```
choose_move(PossibleBoards, Level, FinalBoard):-
    evaluate_moves(PossibleBoards, WinningBoards),
    choose_one_board(Level, PossibleBoards, WinningBoards, FinalBoard).
```

Figura 18 - Predicado que avalia e escolhe a jogada do computador.

O predicado **choose_one_board** está encarregue pela decisão final da jogada. Se o nível de dificuldade escolhido pelo utilizador tiver sido o fácil ("*Easy*"), a jogada será escolhida aleatoriamente de entre todas jogadas válidas possíveis. No entanto, se o nível for médio ("*Medium*"), a jogada será selecionada da lista de jogadas que concluem o jogo. Na eventualidade desta lista não possuir nenhum elemento, a escolha será feita como no modo fácil, isto é, sem critério.

```
choose_one_board(1, PossibleBoards, _, FinalBoard):-
    choose_random_move(PossibleBoards, FinalBoard).

choose_one_board(2, PossibleBoards, WinningBoards, FinalBoard):-
    list_empty(WinningBoards),
    choose_random_move(PossibleBoards, FinalBoard)
    ;
    choose_random_move(WinningBoards, FinalBoard).
```

Figura 19 - Predicado que escolhe a jogada do computador, dependendo da dificuldade escolhida.

Conclusões

A elaboração deste projeto, permitiu-nos abordar, de uma forma prática, conceitos que até à data eram apenas teóricos. Progressivamente, foi-se tornando mais evidente o potencial da linguagem *Prolog* no âmbito da programação lógica e da inteligência artificial.

Ao longo do desenvolvimento deste trabalho, deparámo-nos com algumas dificuldades, principalmente na manipulação de listas, que se revelou um conceito fundamental para a implementação da maioria dos mecanismos do jogo.

É também de salientar, que todos os requisitos pretendidos foram alcançados com sucesso. Porém, existem ainda alguns aspetos que poderiam ser melhorados devido à inexperiência na linguagem, nomeadamente no que diz respeito às capacidades da inteligência artificial implementadas. Teria certamente sido interessante a implementação de algoritmos que habilitassem o computador de prever as jogadas do adversário para que pudesse evitar movimentos desfavoráveis.

Concluímos, assim, que este projeto foi sem dúvida benéfico para a compreensão de um novo paradigma de programação, criando novas possibilidades para aquilo que podemos desenvolver com os conhecimentos recém-adquiridos.

Bibliografia

https://www.boardgamegeek.com/boardgame/286295/quantik