19/20

EXO

RELATÓRIO FINAL – PROGRAMAÇÃO EM LÓGICA

GRUPO EXO_3:

INÊS ALVES – UP201605335@FE.UP.PT

NUNO CARDOSO – UP201706162@FE.UP.PT

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
RUA ROBERTO FRIAS, SN, 4200-465 PORTO, PORTUGAL

17 DE NOVEMBRO DE 2019

Índice

Introdução

O objetivo deste projeto passa por implementar, em linguagem Prolog, um jogo de tabuleiro. O jogo escolhido, “Exo”, foi criado pelo designer francês Léandre Proust cujo tema principal é o espaço. O objetivo passa por criar galáxias e sistemas planetários que giram em torno de uma estrela. Os planetas são diferenciados pelas suas características, relativamente a tamanho, cor e tipo.

O jogo

O jogador que começa o jogo é aquele que mais recentemente olhou para as estrelas do céu - a primeira jogada consiste em colocar uma peça com uma estrela à sua frente. Originalmente, este jogo não tem um tabuleiro definido, contudo decidimos criar um de dimensões 9x9 para cada jogador, de modo a simplificar e a assegurar uma melhor jogabilidade por parte do utilizador

Após ambos os jogadores terem posicionado a sua estrela, têm à sua escolha várias peças, num total de 27. Cada peça representa um planeta e estas estão dispostas de forma aleatória, aglomeradas e podendo estar sobrepostas. Cada jogador, à vez, apenas poderá escolher peças que não estejam cobertas, ou seja, que não tenham outras peças por cima de si. Como este sistema é de difícil implementação, não contribuindo em nada para a jogabilidade, decidimos utilizar três “stacks” para representar montes de peças, cada uma inicialmente com nove peças aleatórias. Assim, o jogador apenas poderá escolher uma das três peças do topo de cada monte, considerando-se as outras como peças cobertas.

Figura 1: Exemplo do jogo na vida real

A peça escolhida é, então, colocada num dos quadrados adjacentes à estrela (na vertical, horizontal ou diagonal). Depois desta primeira jogada, as próximas peças terão de ser colocadas ou, igualmente, adjacentes à estrela, ou adjacentes aos restantes planetas do tabuleiro.

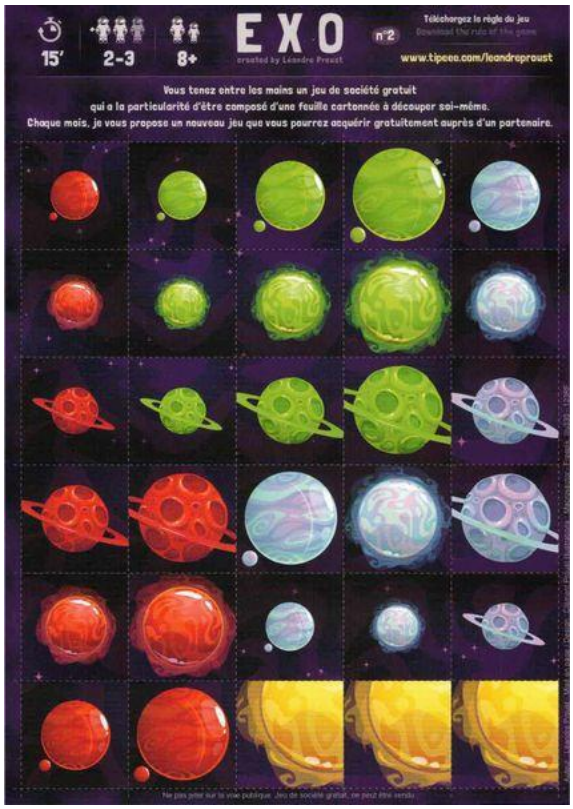

Figura 2: As trinta peças do jogo

Cada planeta tem três características: tamanho (pode ser pequeno, médio ou grande), cor (vermelho, azul, verde) e tipo (terrestre, gasoso, com anéis). Após ser colocada, uma peça não pode ser movida. O objetivo do jogo é, essencialmente, formar grupos de planetas com o maior número de características em comum.

O jogo termina quando ambos os jogadores tiverem 13 planetas na sua galáxia (sobrará sempre uma peça, visto que eram 27 no início do jogo).

Cada jogador ganha um ponto a cada grupo de três planetas que partilhem uma característica. Estes grupos podem ser uma linha de três planetas na vertical, horizontal ou diagonal. Por exemplo, se tivermos uma fila de três planetas com a mesma cor e tipo, a pontuação é dois - um ponto por cada característica partilhada.

Lógica do jogo

Representação do estado do jogo

Tabuleiro

Apesar de no relatório anterior termos definido um tabuleiro de 14x14, decidimos alterá-lo para um 9x9, visto acharmos que ter demasiados espaços torna o jogo confuso e difícil de jogar. Criamos então uma lista de listas, em que cada lista representa uma linha do tabuleiro.

```
init_board(Board1, Board2):-
    Board1 = [ [' 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 ', ' 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 ', ' 0 '],
               [' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 '],
               [' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ']],

    Board2 = [ [' 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 ', ' 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 ', ' 0 '],
               [' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 '],
               [' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ', ' 0 ']].
```

Figura 3: Representação interna dos tabuleiros.

Peças

Para representar as diferentes peças disponíveis para escolha utilizamos 3 listas de 9 peças cada uma. Estas são colocadas a partir de um grupo inicial, **Pieces**, nas três listas (**Pack1**, **Pack2**, **Pack3**) aleatoriamente (detalhes de implementação adiante).

Cada planeta tem três características diferentes, representadas em cada peça por 3 caracteres:

- Um para identificar a cor do planeta: R (vermelho) / G (verde) / B (azul);
- Um para o tamanho: 1 (pequeno) / 2 (médio) / 3 (grande);
- Um para o tipo: X (terrestre) / Y (gasoso) / Z (com anéis).

Por exemplo, a peça g1x representa um planeta verde (g), pequeno (1) e terrestre (x).

Visualização do tabuleiro

Com a função **display_game()** mostramos no ecrã os dois tabuleiros do jogo (chamada a **display_boards()**), legendados com “PLAYER1” e “PLAYER2”, a pontuação atual (chamada a **display_score()**), as peças disponíveis em cada monte para escolha (chamada a **display_stacks()**) e, por fim, qual o jogador que está a jogar (chamada a **display_player_playing()**).
Para iniciar o jogo no terminal, deve-se escrever “**play.**”.

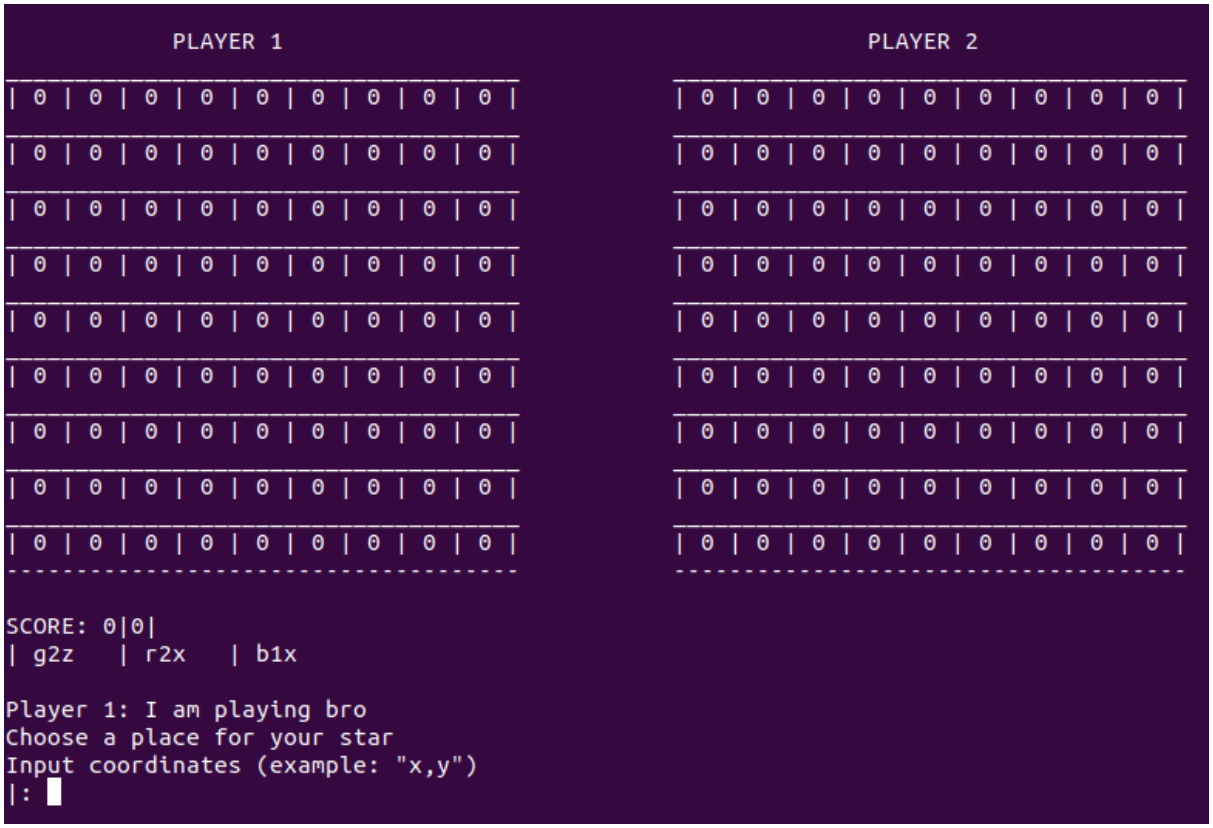

Figura 4: Exemplo de início do jogo

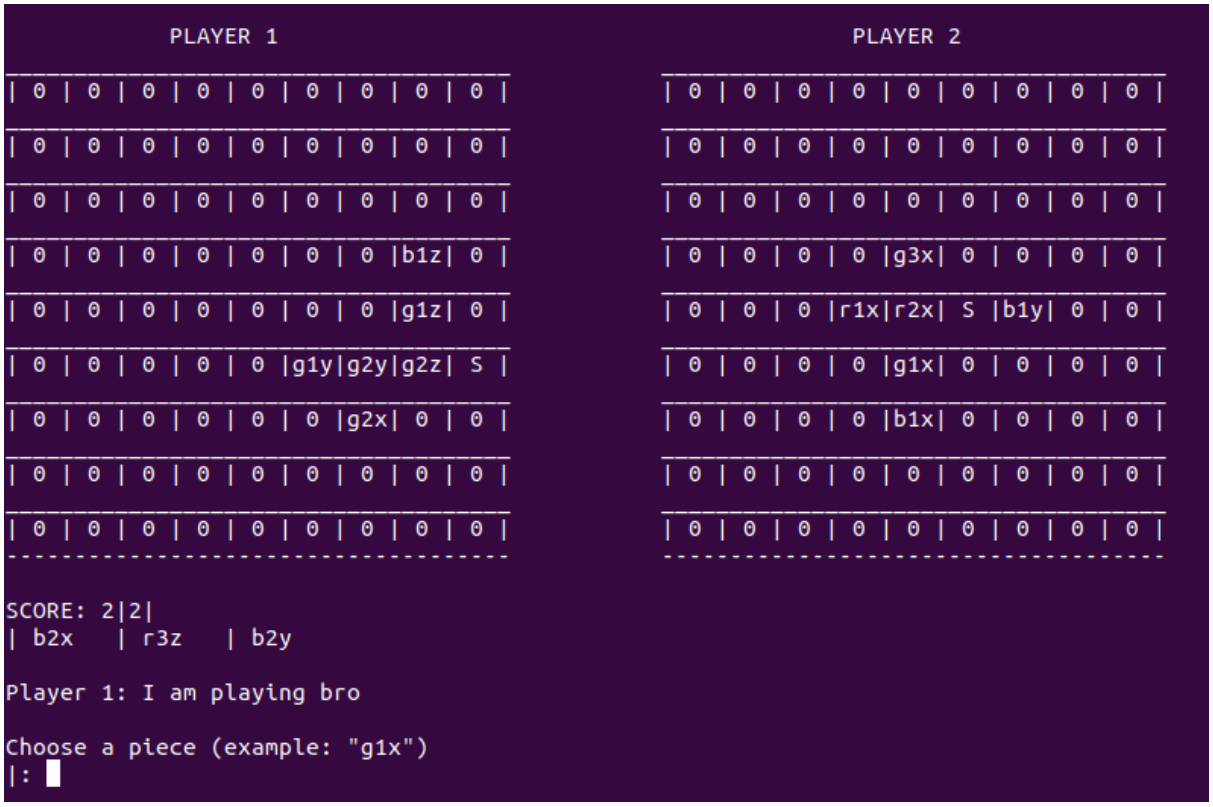

Figura 5: Exemplo de uma jogada intermédia

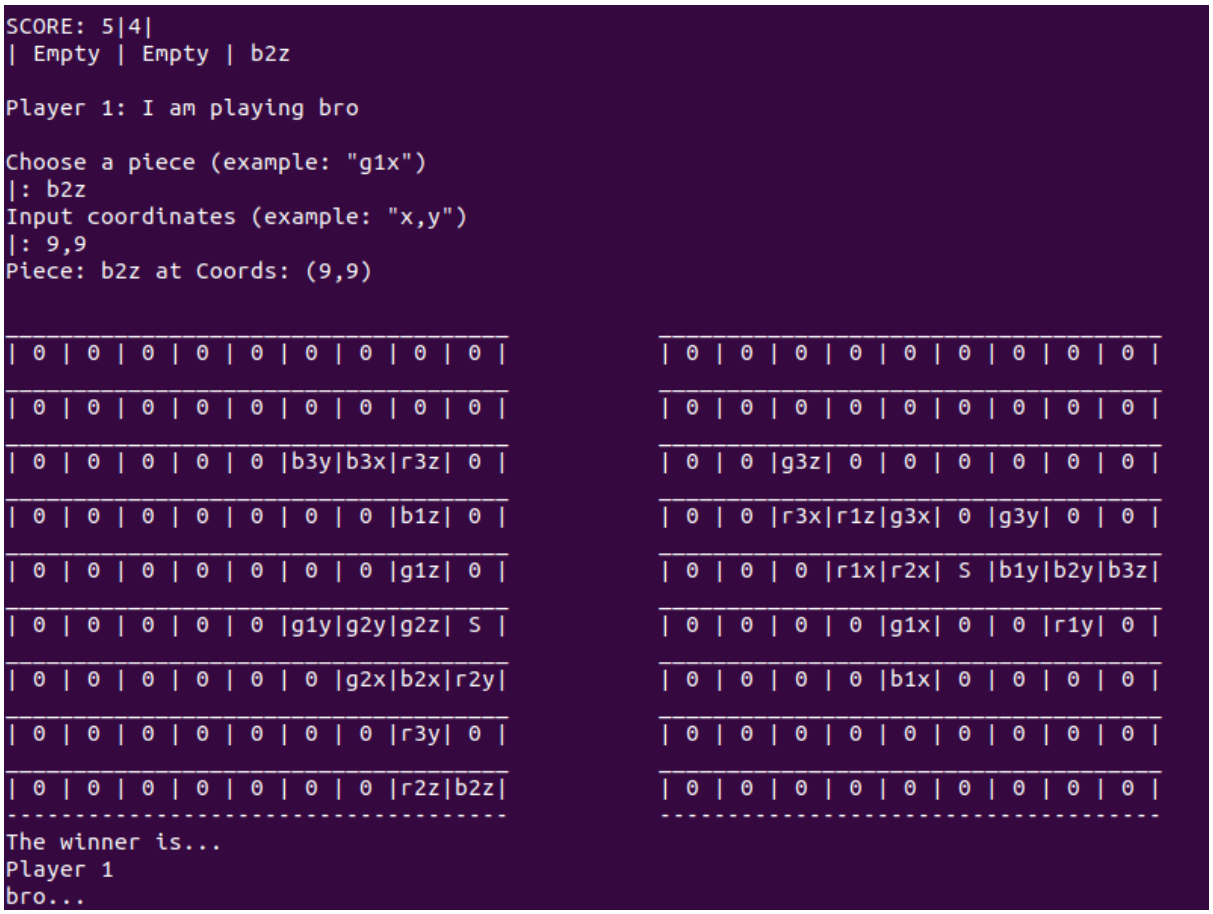

Figura 6: Exemplo de fim do jogo, após colocar uma peça final, com vitória do jogador 1 (5-4)

Lista de jogadas válidas

O predicado **valid_moves()** é utilizado para obter todos as jogadas possíveis para um determinado momento do jogo. Para tal, recebe como valor de entrada uma lista de jogadas possíveis até então, para depois a atualizar. Neste caso, um jogador só poderá colocar uma peça nos espaços adjacentes a uma peça já colocada, ou à estrela do início do jogo - isto é, pode colocar à direita, esquerda, cima, baixo ou na diagonal. Para a posição pretendida, calculamos as coordenadas onde se poderão colocar peças, sendo feitas, depois, chamadas a **update_valid_moves()** para todas as possibilidades.

O predicado **update_valid_moves()** vai, para todas as posições obtidas anteriormente, inicialmente ver se essa posição está vazia (**check_empty_place()** em **get_valid_move()**), e, caso isto se verifique, adicionar essa posição à lista de jogadas possíveis (**add_element()**). Por outro lado, se a posição não estiver vazia, é chamada o predicado **delete_move_from_valid()**, para assegurar que a posição é removida da lista de jogadas disponíveis.

```
%used to get the valid moves for a certain position of the board. A valid move is a position directly on the left, right, up, down or diagonal of a piece or star
valid_moves(Board, InputCoordX, InputCoordY, PreviousPossibleMoves, PossibleMovesOut):-
    XUp is InputCoordX-1,
    XDown is InputCoordX+1,
    X is InputCoordX,
    YLeft is InputCoordY-1,
    YRight is InputCoordY+1,
    Y is InputCoordY,

    update_valid_moves(PreviousPossibleMoves, NewMoves1, Board, XUp, Y),
    update_valid_moves(NewMoves1, NewMoves2, Board, XUp, YRight),
    update_valid_moves(NewMoves2, NewMoves3, Board, XUp, YLeft),
    update_valid_moves(NewMoves3, NewMoves4, Board, XDown, Y),
    update_valid_moves(NewMoves4, NewMoves5, Board, XDown, YRight),
    update_valid_moves(NewMoves5, NewMoves6, Board, XDown, YLeft),
    update_valid_moves(NewMoves6, NewMoves7, Board, X, YRight),
    update_valid_moves(NewMoves7, PossibleMovesOut, Board, X, YLeft).
```

Figura 7: Predicado valid_moves

```
%updates the list of valid moves - checks if a certain move is valid, and, if it is, adds it to the list of moves. otherwise, deletes it
update_valid_moves(PreviousMoves, NewMoves, Board, X, Y):-
    (get_valid_move(Board, X, Y, Move),
    add_element(Move, PreviousMoves, NewMoves));
    delete_move_from_valid(X, Y, PreviousMoves, NewMoves);
    NewMoves = PreviousMoves.

%gets a valid move for a certain position, if that position doesn't have a piece already (check_empty_place)
get_valid_move(Board, X, Y, Move):-
    check_empty_place(Board, X, Y),
    prepare_possible_move(X, Y, Move).

%checks if the position given is empty, that is, if the cell is ' 0 '
check_empty_place(Board, X, Y):-
    get_piece(Board, X, Y, Piece),!,
    Piece == ' 0 '.

%used to format the position to an "X,Y" format
prepare_possible_move(X, Y, Move):-
    string_number(X,XString),
    string_number(Y,YString),
    atom_concat(XString, ',', NewMove),
    atom_concat(NewMove, YString, Move).

%deletes a move from the list of valid moves
delete_move_from_valid(X, Y, PreviousMoves, NewMoves):-
    string_number(X, XString),
    string_number(Y, YString),
    atom_concat(XString, ',', CoordsX),
    atom_concat(CoordsX, YString, Coords),
    delete(PreviousMoves, Coords, NewMoves).
```

Figura 8: Predicados auxiliares usados em valid_moves

Execução de jogadas

Para representar uma jogada, utilizamos um predicado **move()**. Este, tanto para o jogador 1 como para o jogador 2, irá chamar os predicados **set_piece()** e **valid_moves()**:

- **set_piece()** irá colocar a peça escolhida pelo jogador nas coordenadas pretendidas.
- **valid_moves()**, como explicado anteriormente, irá calcular para as coordenadas da peça colocada quais as próximas jogadas possíveis.

```
move(Player, InputCoordX, InputCoordY, InputPiece, Board1, Board2, BoardOut, PossibleMoves1, MovesOut1, PossibleMoves2, MovesOut2):-
    (
        (
            Player = 1,
            set_piece(InputCoordX, InputCoordY, InputPiece, Board1, BoardOut),
            valid_moves(Board1, InputCoordX, InputCoordY, PossibleMoves1, MovesOut1)
        );
        (
            Player = 2,
            set_piece(InputCoordX, InputCoordY, InputPiece, Board2, BoardOut),
            valid_moves(Board2, InputCoordX, InputCoordY, PossibleMoves2, MovesOut2)
        )
    ).
```

Figura 9: Predicado move

Final do jogo

Quando já não há mais peças disponíveis para serem colocadas no tabuleiro (os três grupos de peças - “packs” - estão vazios), o jogo acaba e um vencedor é escolhido. No predicado **game_over()**, chamado então quando os três packs estão vazios (verificação feita no início de **loop()**, como visto na figura 10), dá-se display dos boards (**display_boards()**) e chama-se o predicado **get_winner()**. Este, por sua vez, vai comparar os scores de ambos os jogadores e o vencedor será o que tiver maior score. Em caso de empate, é retornada uma mensagem a dizer o mesmo.

```
loop(Board1, Board2, Score, Pack1, Pack2, Pack3, Player, PossibleMoves1, PossibleMoves2,Choice, PlayerChoice, AI1Level, AI2Level):-
    (Pack1 = [], Pack2 = [], Pack3 = [],
    game_over(Board1, Board2, Score, Winner),
    write('The winner is... '), nl,
    write(Winner), nl,
    write('bro...'), nl);
```

Figura 10: game_over no predicado loop. Dá display do winner obtido

```
get_winner(Score, Winner):-
    nth0(0, Score, Score1),
    nth0(1, Score, Score2),
    ((Score1 > Score2,
    Winner = 'Player 1');
    (Score1 < Score2,
    Winner = 'Player 2');
    Winner = 'Wait! It is a Tie!!! :)').

game_over(Board1, Board2, Score, Winner):-
    display_boards(0, Board1, Board2),
    get_winner(Score, Winner).
```

Figura 11: Predicados game_over e get_winner

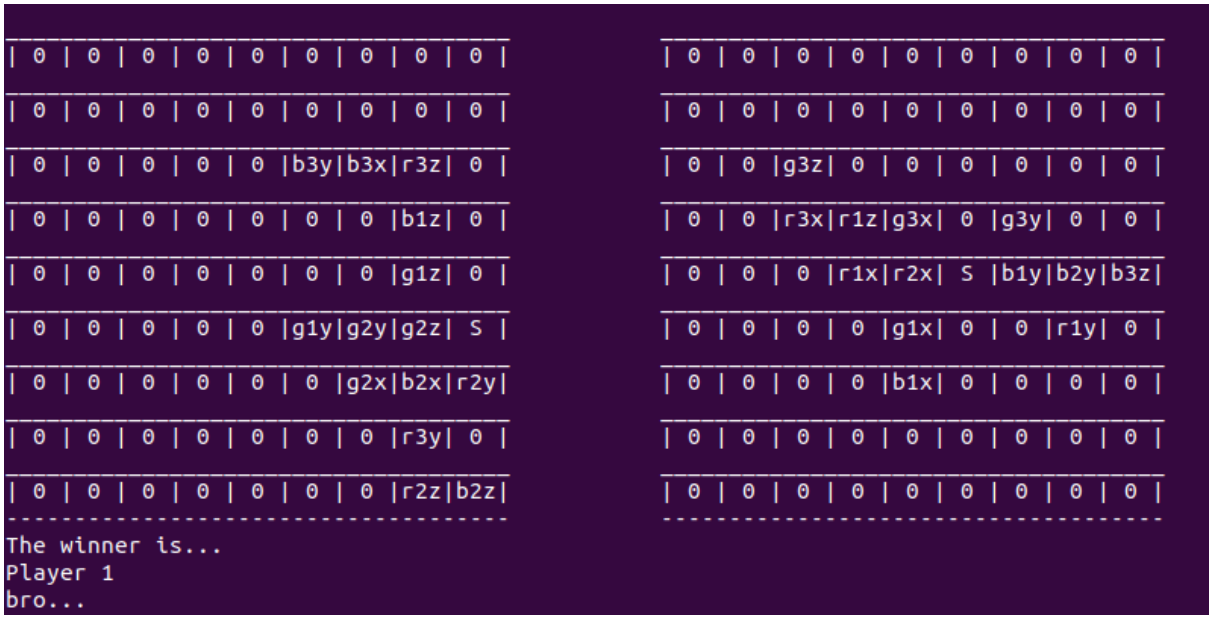

Figura 12: Exemplo de fim do jogo, com display dos boards e do jogador que ganhou

Avaliação do tabuleiro

De modo a obter uma boa avaliação do estado do tabuleiro, de modo a decidir a melhor jogada por parte do computador, utilizamos o predicado **value()** e seus auxiliares. O predicado **value()** limita-se a indicar qual a pontuação resultante de uma determinada jogada hipotética. Assim, o predicado **get_best_value()** serve-se dele para calcular recursivamente a posição onde a jogada fará obter uma pontuação mais elevada. Por sua vez, o predicado **get_best_move_for_piece()**, que recebe como valor de entrada a peça a ser usada na jogada, recorre ao **get_best_value()**, para escolher qual a melhor posição onde colocar a peça. Por fim, o predicado **get_best_move()** executa o predicado **get_best_move_for_piece()** com todas as peças disponíveis para jogar e obtém qual a melhor jogada a ser executada pelo computador: melhor peça a ser escolhida e a ser colocada no local que dará mais pontuação. Caso não exista nenhuma jogada a fazer obter mais pontuação, o computador limita-se a colocar a primeira peça livre, num lugar aleatório do tabuleiro que corresponda a uma jogada possível.

```
%to get the score (Value) of a specific move
value(Board, X, Y, PossibleMoves, Piece, Player, Value):-
    move(Player, X, Y, Piece, Board, Board, _, PossibleMoves, _, PossibleMoves, _),
    verify_combination(Board, X, Y, Piece, 0, Value).
```
Figura 13: Predicado value

```
%used to find the best moves for the AI level 2
get_best_move(Board, Pack1, Pack2, Pack3, PackUsed, PossibleMoves, Player, BestX, BestY, BestPiece):-
    (nth0(0, Pack1, Piece1);Piece1=' 0 '),
    (nth0(0, Pack2, Piece2);Piece2=' 0 '),
    (nth0(0, Pack3, Piece3);Piece3=' 0 '),
    get_best_move_for_piece(Board, PossibleMoves, Piece1, Player, 0, _, _, Best1X, Best1Y, Best1Value),!,
    get_best_move_for_piece(Board, PossibleMoves, Piece2, Player, 0, _, _, Best2X, Best2Y, Best2Value),!,
    get_best_move_for_piece(Board, PossibleMoves, Piece3, Player, 0, _, _, Best3X, Best3Y, Best3Value),!,
    (
        (Best1Value > Best2Value, Best1Value > Best3Value, BestX = Best1X, BestY = Best1Y, BestPiece = Piece1, PackUsed = 1);
        (Best2Value > Best1Value, Best2Value > Best3Value, BestX = Best2X, BestY = Best2Y, BestPiece = Piece2, PackUsed = 2);
        (Best3Value > Best2Value, Best3Value > Best1Value, BestX = Best3X, BestY = Best3Y, BestPiece = Piece3, PackUsed = 3)
    ).

%getting the best possible move for a certain piece
get_best_move_for_piece(Board, PossibleMoves, Piece, Player, InitialValue, InitialX, InitialY, BestX, BestY, BestValue):-
    length(PossibleMoves, N),
    get_best_value(N, Board, PossibleMoves, PossibleMoves, Piece, Player, InitialValue, InitialX, InitialY, BestX, BestY, BestValue).

%recursively checks what move will bring the most points to the AI
get_best_value(1, _, _, _, _, _, Value, X, Y, BestX, BestY, BestValue):-
    BestX = X,
    BestY = Y,
    BestValue = Value.

get_best_value(N, Board, PossibleMoves, [PossibleMove|Tail], Piece, Player, InitialValue, InitialX, InitialY, BestX, BestY, BestValue):-
    Next is N-1,
    atom_chars(PossibleMove, ListMove),
    get_coord(0, ListMove, X),
    get_coord(2, ListMove, Y),
    value(Board, X, Y, PossibleMoves, Piece, Player, Points),
    ((Points > InitialValue, NewValue = Points, NewX = X, NewY = Y);
    (NewValue = InitialValue, NewX = InitialX, NewY = InitialY)),
    get_best_value(Next, Board, PossibleMoves, Tail, Piece, Player, NewValue, NewX, NewY, BestX, BestY, BestValue).
```

Figura 14: Predicados utilizados para calcular as melhores jogadas para a AI

Jogada do computador

As jogadas do computador são escolhidas pelo predicado **choose_move()** e executadas pelo normal funcionamento do jogo. Estas dividem-se em dois sentidos:

- caso o nível escolhido corresponda ao primeiro nível, o computador irá escolher aleatoriamente um baralho de entre os 3 disponíveis e jogar a peça do seu topo num local aleatório do tabuleiro, que corresponda a uma jogada válida (posição adjacente a outra peça já colocada no tabuleiro);
- caso o nível escolhido corresponda ao segundo nível, o computador irá executar os mecanismos mencionados anteriormente, na secção “Avaliação do Tabuleiro”, para calcular qual a jogada a dar mais pontuação, ou no caso de não haver aumento da pontuação irá fazer o mesmo que o primeiro nível.

```
%handles the moves for the AI according to the difficulty level (AILevel)
choose_move(Board, Pack1, Pack2, Pack3, PossibleMoves, AILevel, InputCoordX, InputCoordY, PackUsed, Piece, MoveType):-
    %for AI level 1, the moves are randomly chosen from a list of possible moves
    (AILevel = 1,
        (MoveType = 'coords',
        (((\+var(PossibleMoves)),
        length(PossibleMoves, Size),
        randomize_number(Size, MoveNr),
        nth0(MoveNr, PossibleMoves, PossibleMove),
        atom_chars(PossibleMove, Move),
        get_coord(0, Move, InputCoordX),
        get_coord(2, Move, InputCoordY));
        (var(PossibleMoves),
        random(1, 10, InputCoordX),
        random(1, 10, InputCoordY)))
        ;
        (MoveType = 'piece',
        repeat,
        random(1, 4, PackUsed),
        ((PackUsed = 1, nth0(0, Pack1, Piece));
        (PackUsed = 2, nth0(0, Pack2, Piece));
        (PackUsed = 3, nth0(0, Pack3, Piece))),!,
        choose_move(Board, Pack1, Pack2, Pack3, PossibleMoves, AILevel, InputCoordX, InputCoordY, PackUsed, Piece, 'coords')))
    );
    %for AI level 2, the moves are chosen according to what moves give the AI the best score
    (AILevel = 2,
        ((MoveType = 'coords',
        choose_move(Board, Pack1, Pack2, Pack3, PossibleMoves, 1, InputCoordX, InputCoordY, PackUsed, Piece, 'coords'))
        ;
        (MoveType = 'piece',
            (get_best_move(Board, Pack1, Pack2, Pack3, PackUsed, PossibleMoves, _, InputCoordX, InputCoordY, Piece)
            ;
            (choose_move(Board, Pack1, Pack2, Pack3, PossibleMoves, 1, InputCoordX, InputCoordY, PackUsed, Piece, 'piece')))))
    ).
```

Figura 15: Predicado choose_move

Conclusões

Este trabalho ajudou-nos a consolidar vários conceitos dados da linguagem de programação Prolog, linguagem com a qual não estávamos familiarizados até agora. Apesar de acharmos que o jogo de tabuleiro escolhido não foi o melhor, visto ter regras confusas e nos dificultar a implementação, consideramos que conseguimos cumprir todos os objetivos com sucesso.

Bibliografia

Regras do jogo:

https://boardgamegeek.com/boardgame/276489/exo

Prolog:

Sterling, Leon; The Art of Prolog.

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