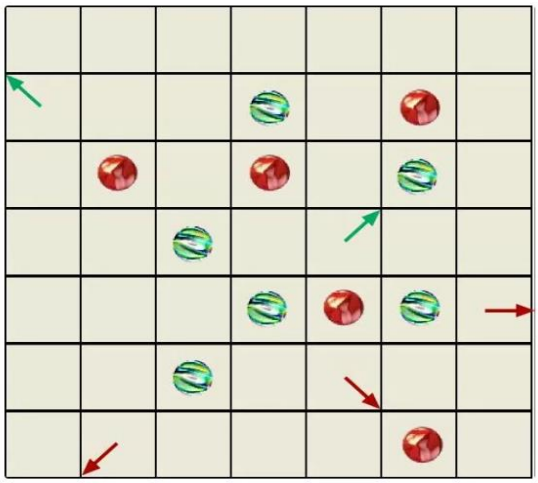
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Faculdade de Engenharia da Universidade do Porto

Momentum

Trabalho Prático 1



Programação Funcional e em Lógica

Turma 2 – Momentum 7

André Tiago Moreira Soares da Costa e Silva (up202108724@fe.up.pt)
Contribuição:50%

João Tomás Cardinal Pinho Teixeira (up202108738@fe.up.pt). Contribuição: 50%

Porto, 5 de novembro de 2023

Índice

Instalação e Execução

Para instalar e executar o jogo Momentum, é necessário fazer o download dos ficheiros presentes em PFL_TP1_T02_Momentum7.zip e descompactá-los. Os ficheiros fonte do jogo encontram-se na pasta src, pelo que é necessário consultar o ficheiro menu.pl através da UI do SICSTUS Prolog 4.8.0. O jogo está disponível em ambientes Windows e Linux.

O jogo inicia-se com o predicado play/0:

? – play.

Descrição do Jogo

Momentum é um jogo de tabuleiros para dois jogadores, que, na sua versão tradicional tem lugar num tabuleiro 7x7. Inicialmente vazio, os jogadores detêm inicialmente 8 peças (também conhecidas como “marbles”), e ganha quem conseguir colocar primeiro todas as suas peças no tabuleiro. Contudo, é necessário tem em conta certas regras que definem a dinâmica dos movimentos que acontecem no jogo:

- Um jogador só poderá colocar uma peça por turno.
- Dá se um “momentum” quando um jogador coloca uma peça adjacente a pelo menos uma peça já presente no tabuleiro, “empurrando” a mesma peça uma casa para a direção em que se formou a adjacência (isto é, se se colocar uma peça à esquerda de uma peça já existente, a peça já existente avançará uma casa para a direita (se esta estiver disponível).
- Os “momentums” podem ocorrer em todas as 8 direções, e em várias direções em simultâneo.
- No caso excecional de se empurrar uma linha, apenas a última peça na direção da linha será empurrada, pelo que as restantes as peças desta linha manter-se-ão imóveis.
- Caso a peça empurrada não tenha um destino nos limites do tabuleiro, ela sai do tabuleiro e volta à mão do jogador respetivo, podendo ser jogada posteriormente.

Também é necessário ter em conta outras regras que não definem o movimento das peças, mas influenciam o fluxo de jogo:

- No primeiro turno do segundo jogador exclusivamente, o primeiro jogador já terá colocado a sua peça, pelo que o segundo jogador poderá trocar a peça colocada por uma das suas peças, sendo devolvida a peça trocada ao jogador respetivo.
- Após o primeiro jogador colocar a sua penúltima peça, este deverá anunciar que apenas terá uma peça. Assim, o jogador seguinte será forçado a empurrar, se possível, uma peça adversária para fora do tabuleiro, caso contrário, perderá.
- O jogo tem um limite de 60 turnos (30 turnos para cada jogador). Neste caso em particular, o jogador vencedor é o jogador que detêm mais peças em campo. Excecionalmente, se o número de peças de cada jogador continuar a ser igual, declarar-se-á um empate.

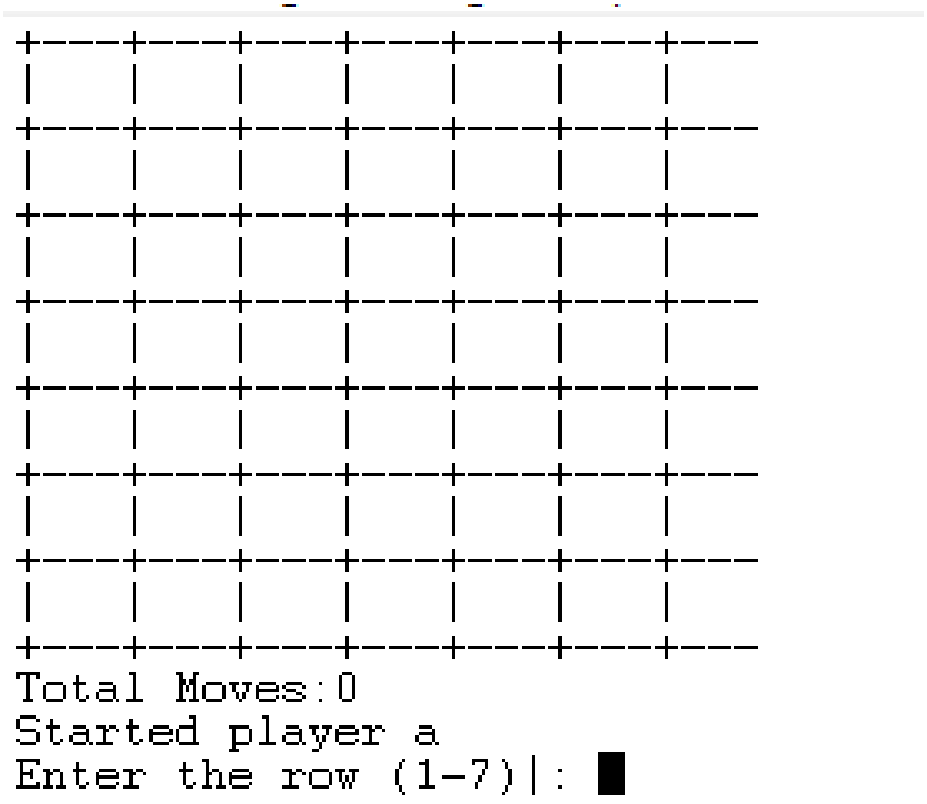
Lógica do Jogo

Representação do estado de jogo

O estado de jogo (denominado de “Game State”), é um argumento essencial que tem influência nos predicados de mais alto nível da implementação. É formado por uma lista de quatro elementos:

- Board. Uma matriz (representada por um array bi-dimensional), de tamanho indicado pelo utilizador no momento da configuração. Cada célula é representada por um valor atómico, sendo o vazio representado como ‘empty’ e desenhado como ‘_’, e o primeiro jogador é representado por um ‘player1’, desenhado como ‘P1’, e o segundo jogador representado de forma semelhante, correspondendo ao valor atómico ‘player2’ e desenhado como ‘P2’.
- Player: jogador representante do turno atual (átomo player1 ou player2);
- MarblesOnBoard: Uma lista de coordenadas de todas as peças de cada jogador presentes atualmente no tabuleiro ([player1, 2, 3], [player2,4,5], por exemplo, corresponderia a um tabuleiro no qual o primeiro jogador terá uma peça na posição 2,3 e o player2 terá uma peça na posição 4,5).
- TotalMoves, acumulador do número total de turnos/movimentos durante o jogo. Com este valor foi possível limitar jogos com muitos movimentos (o que é recorrente em jogos amadores).

Representação do estado inicial do jogo:



Representação do estado final do jogo:

```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   | P1|   | P1| P2|   |   |
+---+---+---+---+---+---+---+
|   |   |   | P2|   | P1|   |
+---+---+---+---+---+---+---+
| P2| P1| P1|   |   |   |   |
+---+---+---+---+---+---+---+
|   | P1|   | P1|   | P2|   |
+---+---+---+---+---+---+---+
|   |   |   | P1|   | P2|   |
+---+---+---+---+---+---+---+
|   |   |   | P2|   |   |   |
+---+---+---+---+---+---+---+
Total Moves:41
The game is over! The winner is bot1
Clearing data
Cleared data
true ? yes
| ?-
```

Visualização do estado de jogo

Antes de iniciar o jogo, o(s) jogador(es) configuram a partida, construindo também o estado inicial. Os parâmetros a serem configurados são:

- Modo (Humano/Humano, Humano/Computador, Computador/Computador)
- Nome dos jogadores (se aplicável)
- O jogador que inicia a partida
- O tamanho do tabuleiro escolhido onde começar o jogo.

A validação de inputs é garantida, e todos os estados iniciais são criados com qualquer combinação de configurações possível.

Uma possível interação é a seguinte:

```
% consulted c:/users/andre silva/desktop/pfl_tpl_tU2_#group/menu.pl in module u
ser, 0 msec 592336 bytes
| ?- play.
================================================

    ( M )( O )( M )( E )( N )( T )( U )( M )

================================================
Game modes:
1 - Human vs. Human
2 - Human vs. Computer
3 - Computer vs. Computer
Select a mode between 1 and 3: 1.
Human vs. Human
Enter the name: |: 'André'.
Enter the name: |: 'João'.
Enter the board size (minimum 7): |: 7.
Who starts playing?
1 - André blue
2 - João red
Select between 1 and 2: |: 1.
```

Neste exemplo em particular, escolheu-se o modo Humano/Humano, tratando-se de um duelo entre dois jogadores, que posteriormente escolhem os seus nomes, seguidamente escolhendo o tamanho do tabuleiro (neste caso particular escolheu-se o tabuleiro normal (o 7x7).

Como também é visível no exemplo mostrado acima, são avaliadas 2 opções (a de modo de jogo, e o jogador que começa a jogar primeiro). Assim, a validação da escolha destas opções é feita com o uso do predicado get_option(+Min,+Max.+Context,-Value), em que Min é a opção menor a ser escolhida, Max é a opção maior a ser escolhida, o contexto é um placeholder a ser usado na string que introduz a capacidade de escolha, sendo o Value a opção escolhida:

```
% get_option(+Min, +Max, +Context, -Value)
% Get a valid option from the user
get_option(Min, Max, Context, Value):-
    format('~a between ~d and ~d: ', [Context, Min, Max]),
    repeat,
    (
        read_number(Value),
        (between(Min, Max, Value) -> true ;
        write('Invalid input. Please enter a valid number between '), write(Min), write(' and '), write(Max), nl,
        fail)
    ).
```

Após escolher o modo de jogo, são pedidos os nomes dos jogadores. Assim, foi concebida o get_name(-Player), que inquere sobre os nomes pelos quais os jogadores querem ser referidos. Infelizmente, não foram ultrapassadas as limitações de Prolog, assim, para escolher um nome que use letras maísculas, acentos e outros caracteres especiais, é necessário colocar o nome pretendido entre plicas ‘’. Caso contrário o nome irá falhar, mostrando a mensagem de erro evidenciada pela chamada do write.

```
% get_name(-Player)
% Get the name of a player
get_name(Player) :-
    write('Enter the name: '),
    read(Name),
    (\+ is_bot_name(Name),atom(Name) ->
        asserta(name_of(Player, Name))
    ;
        write('The name is not valid. Please choose a different name.\n'),
        get_name(Player)
    ).
```

Após a escolha do tamanho do tabuleiro, este vai ser inicializado com initial_state(+size, -Board), em que Board é um array bi-dimensional com size listas, cada uma preenchida com size elementos . É de notar que o recomendado é o tabuleiro 7x7, uma vez que no caso de o tabuleiro ser muito grande, haverá muita variedade de posições a serem jogadas, sendo mais difícil garantir uma dinâmica adequada de jogo (sendo cada vez mais vantajoso para o jogador que por a primeira peça).

Assim, concluem-se as configurações de jogo, que inicializam o primeiro GameState:

```
% game_configurations(-GameState)
% Set the game configurations
game_configurations([Board,Player,[],0]):-
        header,
        set_mode,
        choose_board_size(7, Size),
        initial_state(Size,Board),
        initialize_marbles([]),
        choose_player(Player),
        init_random_state,
        assertz(board_size(Size)).
```

Na inicialização do game state, bem como em cada mudança deste, o tabuleiro é mostrado, recorrendo ao predicado display_state(+GameState). Assim, o display_state só usará o elemento Board do estado de jogo (para o mostrar), e as TotalMoves (para mostrar o número de jogadas feitas até ao momento). Também, é importante salientar que antes de executar a sua ação, esta limpa o IDE (a consola), “refrescando” o tabuleiro (com o predicado clear_console/0).

```
% display_state(+GameState)
% Display the current state of the game
display_state(GameState):-
    clear_console,
    [Board,_,_,TotalMoves]= GameState,
    print_board(Board), nl,
    format('Total Moves:~d', [TotalMoves]), nl.
```

Execução das jogadas:

O jogo utiliza um ciclo que termina quando um jogador vence o jogo, ou seja, a condição de vitória é cumprida.

```
% game_cycle(-GameState)
% Game cycle that ends when the game is over
game_cycle(GameState):-
    [_, Player,_, _] = GameState,
    game_over(GameState,Player),!,
    winner(Winner),
    name_of(Winner,NameofWinner),
    format('The game is over! The winner is ~w~n', [NameofWinner]).
game_cycle(GameState):-
    [Board,Player,X, NewTotalMoves]=GameState,
    name_of(Player, NameofPlayer),
    format('Started player ~w~n', [NameofPlayer]),
    move(GameState, NewGameState),
    game_cycle(NewGameState).
```

O jogador escolha a posição onde pretende colocar a sua peça através do predicado choose_position/2. Neste, o jogador introduz a linha e a coluna da posição pretendida e a peça é colocada através do predicado place_marble/3 que verifica se essa posição está livre. Caso a peça não poder ser colocada, quando o jogador não se encontra a perder, serão pedidas novas coordenadas ao jogador, até que o jogador escolha uma posição válida.

```
choose_position(Player,TotalMoves):-
    board_size(Size),
    format('Enter the row (1-~d)', [Size]),
    read_number(Row),
    format('Enter the column (1-~d)', [Size]),
    read_number(Column),
    (place_marble(Player, Row, Column) ->
        true;
        TotalMoves = 1 ->
        replace_marble(Player, Row, Column)
    ;
        write('Invalid position. Please choose a valid position.\n'),
        choose_position(Player,TotalMoves)
    ).
```

Contudo, em casos excecionais, em que existe um diferencial de vantagem, o jogador em desvantagem terá um conjunto mais limitado onde poderá colocar a sua peça/fazer a sua jogada, pelo que as suas jogadas serão invalidadas enquanto não jogar forçosamente numa posição desse conjunto. Essa verificação é feita também recursivamente com o predicado choose_forced_position(+Player, +ForcedMoves), que verifica se a linha e a coluna escolhidas pertecem ao ForcedMoves. Caso não pertencer, serão rejeitadas, tal como as linhas e colunas não existentes no tabuleiro, pelo que serão inquiridas novas coordenadas.

```
% choose_forced_position(+Player,+ForcedMoves)
% Choose one of the forced positions to place a marble
choose_forced_position(Player,ForcedMoves):-
    board_size(Size),
    write('Be careful, you are about to lose'),
    format('Enter the row (1-~d)', [Size]),
    read_number(Row),
    format('Enter the column (1-~d)', [Size]),
    read_number(Column),
    (member((Row,Column),ForcedMoves)->place_marble(Player, Row, Column);
        write('Invalid position. Please choose a valid position.\n'),
        choose_forced_position(Player,ForcedMoves)
    ).
```

O predicado move/2 atualiza o GameState para a próxima jogada construindo o NewGameState, que será usado no próximo game_cycle/1, e também verifica se o jogador é um bot. No caso de ser um bot, este coloca a peça num local aleatório através de uma abordagem random, senão, caso seja um jogador, o predicado choose_position/2 é invocado, sendo um jogador (humano) a colocar a peça. Assim, a fim de conservar a mesma board de GameState para GameState, também é no move que se retiram as coordenadas das peças anteriormente presentes no tabuleiro (usando o MarblesOnBoard), e colocam-se as peças que estarão presentes no tabuleiro do NewGameState, após a peça ser colocada e se darem as transferências devidas.

```
% move(GameState, NewGameState)
% Game action that builds a new GameState, representing a new move on the game

move(GameState, NewGameState) :-
    [Board, Player, MarblesOnBoard, TotalMoves] = GameState,
    erasing_old_coordinates(Board, MarblesOnBoard, ErasedBoard),
    board_size(Size),
    forced_moves(Player, Size, MarblesOnBoard, ForcedMoves),
    length(ForcedMoves, NumForcedMoves),
    (is_bot(Player) ->
        (NumForcedMoves is 0 ->
            generate_all_coordinates(Size, Coordinates),
            filter_available_moves(Coordinates, MarblesOnBoard, AvailableMoves),
            random_member((Row, Column), AvailableMoves),
            NewTotalMoves is TotalMoves + 1,
            place_marble(Player, Row, Column)
        ;
            random_member((Row, Column), ForcedMoves),
            place_marble(Player, Row, Column),
            NewTotalMoves is TotalMoves + 1
        )
    ;
        (NumForcedMoves is 0 ->
            choose_position(Player, TotalMoves),
            NewTotalMoves is TotalMoves + 1
        ;
        write('Forced Moves: '),
        print_list(ForcedMoves),
        choose_forced_position(Player,ForcedMoves),
        NewTotalMoves is TotalMoves + 1
        )
    ),
    change_player(Player, NewPlayer),
    marbles_on_board(X),
    update_board_with_new_coordinates(ErasedBoard, X, NewBoard),
    display_state([NewBoard, _, _, NewTotalMoves]),
    NewGameState = [NewBoard, NewPlayer, X, NewTotalMoves].
```

Lista de jogadas válidas:

No caso do jogo Momentum, as jogadas válidas serão todas as coordenadas que estiverem livres, no entanto, e indo de encontro as regras do jogo, quando um jogador está prestes a perder o jogo este é forçado a empurrar uma peça do adversário para fora do tabuleiro, o mesmo também acontece para o jogo com/entre bots.

Assim, embora não se ter implementado diretamente o predicado valid_moves(+GameState, +Player, -ListOfMoves), este expressa-se no contexto de uma move(+GameState,-NewGameState), com o uso de três predicados fundamentais: o generateCoordinates(+Size,-Coordinates), que receberá o tamanho do tabuleiro (que advém do predicado dinâmico board_size(Size), cujo valor é definido nas configurações do jogo), e retornará, com o uso do findAll(), uma lista de todos os pares de coordenadas existentes no tabuleiro. Seguidamente, com o auxílio da lista MarblesOnBoard, avaliam-se as peças já pertencentes ao tabuleiro (usando o predicado auxiliar coord_taken), e por fim , no filter_available_marbles, usa-se em conjunto ambos estes predicados para gerar uma lista de posições não ocupadas(AvailableMoves), isto é, posições nas quais uma peça poderá ser colocada. É de notar que elementos como a lista MarblesOnBoard provêm do GameState Atual.

```
% generate_all_coordinates(+Size, -Coordinates)
% Generate all the coordinates of the board
generate_all_coordinates(Size,Coordinates):-
    findall((X,Y),(between(1,Size,X),between(1,Size,Y)),Coordinates).

% coord_taken(+MarblesOnBoard, +Coordinate)
% Check if a coordinate is taken by a marble
coord_taken(MarblesOnBoard,(X,Y)):-
    member((_,X,Y), MarblesOnBoard).

% filter_available_moves(+Coordinates, +MarblesOnBoard, -AvailableMoves)
% Filter the coordinates that are not taken by a marble
filter_available_moves(Coordinates, MarblesOnBoard, AvailableMoves):-
    exclude(coord_taken(MarblesOnBoard), Coordinates, AvailableMoves).
```

Final do jogo:

No ciclo do jogo, através do predicado game_over(+GameState, -Winner) são verificadas as condições para o fim do jogo. A condição de vitória é verificada, ora quando um jogador tem 8 peças no tabuleiro (com o auxílio do predicado has_won_game(+Player,+MarblesOnBoard), ora em casos excecionais quando o número total de jogadas/turnos chega às 60, o jogador com mais peças no tabuleiro é declarado vencedor, caso contrário, verifica-se um empate entre os dois. Para efeitos práticos, a verificação da condição de vitória está a ser feita no turno do seguinte à jogada que garantiu vitória ao jogador anterior.

```
% game_over(+GameState, -Winner)
% Predicate to check if the game is over
game_over(GameState, Winner) :-
    [_,_,MarblesOnBoard,_]= GameState,
    change_player(Winner, Opponent),
    has_won_game(Opponent, MarblesOnBoard),!,
    assertz(winner(Opponent)).
game_over(GameState,Winner):-
    [_,Player,MarblesOnBoard,TotalMoves]=GameState,
    TotalMoves is 60,
    change_player(Player,OtherPlayer),
    findall((Player, _, _), member((Player, _, _), MarblesOnBoard), Player1Marbles),
    findall((OtherPlayer, _, _), member((NewPlayer, _, _), MarblesOnBoard), Player2Marbles),
    length(Player1Marbles, NumMarbles1),
    length(Player2Marbles, NumMarbles2),
    (NumMarbles1 > NumMarbles2 -> true ,
    assertz(winner(Player));
    NumMarbles2 > NumMarbles1 -> true, assertz(winner(OtherPlayer));
    assertz(winner('Draw'))).
```

Avaliação do Estado de Jogo

O Estado de Jogo é fundamentalmente avaliado quando um jogador se encontra prestes a ganhar, com sete peças no tabuleiro, e quando o jogador perde esta posição de vantagem. Assim, seguindo as regras do jogo previamente explicitas, o segundo jogador necessitará de forçar uma jogada a fim de evitar a derrota, criando-se uma lista de posições forçadas a serem jogadas. Para o mostrar, temos então o is_player_winning(+Player,+MarblesOnBoard), que , usando a lista MarblesOnBoard do estado atual, avalia a hipótese de o jogador ganhar o jogo no seu próximo turno, e o has_not_winning_anymore(Player,MarblesonBoard), que usando a mesma lista, verifica se o jogador não se encontra numa posição vantajosa.

```
% has_not_winning_anymore(+Player, +MarblesOnBoard)
% Predicate to check if the player is not winning the game
has_not_winning_anymore(Player, MarblesOnBoard) :-
    findall((Player, _, _), member((Player, _, _), MarblesOnBoard), PlayerMarbles),
    length(PlayerMarbles, NumMarbles),
    NumMarbles =< 6.

% is_player_winning(+Player, +MarblesOnBoard)
% Predicate to check if the player is winning the game
is_player_winning(Player, MarblesOnBoard) :-
    findall((Player, _, _), member((Player, _, _), MarblesOnBoard), PlayerMarbles),
    length(PlayerMarbles, NumMarbles),
    NumMarbles is 7.
```

Com a implementação destas funções, foi possível avaliar a lista de jogadas forçadas, isto é, jogadas feitas por um jogador que tirarão a vantagem do jogador adversário. Assim, criou-se o predicado forced_moves(+Player,+Size,+MarblesOnBoard,-ForcedMoves), que usa estes predicados para encontrar o conjunto de moves para o jogador do estado atual, num tabuleiro de tamanho (Size), tendo em conta as peças já jogadas, verificar o conjunto de coordenadas, que, em uma das quais jogada uma peça, tiram a vantagem ao jogador adversário:

```
% Predicate to get all the forced moves for a player where it push the opponent marble to out of the board
forced_moves(Player,_,MarblesOnBoard,[]):-
change_player(Player,NewPlayer),
\+ is_player_winning(Player,MarblesOnBoard),
\+ is_player_winning(NewPlayer,MarblesOnBoard).

forced_moves(Player,Size,MarblesOnBoard,ForcedMoves):-
    generate_all_coordinates(Size,Coordinates),
    change_player(Player,Opponent),
    filter_available_moves(Coordinates, MarblesOnBoard, AvailableMoves),
    findall((X, Y), (

        member((X,Y), AvailableMoves),
        \+ is_marble_at(_,X,Y,MarblesOnBoard),
        % Simulate placing the marble on the board
        simulate_move(MarblesOnBoard, (Player, X, Y), NewMarblesOnBoard),
        % Check if the opponent would win if they placed a marble at (X, Y)
        has_not_winning_anymore(Opponent, NewMarblesOnBoard)
    ), ForcedMoves).
```

Jogada do computador:

Com o objetivo de implementar o bot aleatório, usamos a livraria “random”, a fim de se implementar o predicado random_member() que escolhe de forma aleatória da lista de posições disponíveis uma posição para colocar a peça. Para o caso das posições forçadas, o random_member também escolhe de entre as posições obrigatórias uma posição aleatória entre elas. Também é importante salientar o import do predicado now(X) da livraria “system”, que obterá o instante atual do sistema (current system time), sobre a qual, é criada uma seed(com o método setrand() , para garantir independência da execução de predicados aleatórios para qualquer instância da execução do jogo. Assim, foi possível implementar o init_random_state/0, que, sendo instanciado no início da execução, garante que o bot usará movimentos totalmente aleatórios.

```
(is_bot(Player) ->
    (NumForcedMoves is 0 ->
        generate_all_coordinates(Size, Coordinates),
        filter_available_moves(Coordinates, MarblesOnBoard, AvailableMoves),
        random_member((Row, Column), AvailableMoves),
        NewTotalMoves is TotalMoves + 1,
        place_marble(Player, Row, Column)
    ;
        random_member((Row, Column), ForcedMoves),
        place_marble(Player, Row, Column),
        NewTotalMoves is TotalMoves + 1
    )
;
```

```
% init_random_state/0:
% Initializes the random number generator with a seed based on the current system time.
init_random_state:-
    now(X),
    setrand(X).
```

Conclusões

O jogo Momentum foi implementado com sucesso em Prolog. É possível ser jogado por dois jogadores, por um jogador e por um bot, ou simular um jogo entre dois bots. Contudo, apenas foi possível implementar uma dificuldade para os bots, isto é, a dificuldade Random, uma vez que a grande dificuldade do desenvolvimento residiu na implementação das regras de dinâmica de peças, às quais se veio acrescentar a obtenção de jogadas forçadas que obrigou a uma avaliação de todos os tabuleiros possíveis após a colocação de uma peça. Por fim, considera-se que os conhecimentos adquiridos nas aulas teóricas e práticas foram consolidados, uma vez que o projeto exigiu um domínio mais complexo dos mesmos.

Bibliografia:

https://boardgamegeek.com/boardgame/73091/momentum

https://www.iggamecenter.com/en/rules/momentum