Mapello

Turma 1 Mapello_5

Trabalho realizado por:

Nome	Número
Pedro Miguel Oliveira Azevedo	201603816
Diogo Ferreira de Sousa	201706409

Instalação e execução

Extrair os conteúdos do zip e usando o SICStus fazer a consulta do ficheiro game.pl .

Para executar o jogo é necessário executar o predicado play/0. Após executar o predicado pode escolher entre 3 opções :

- Player vs Player
- Player vs CPU
- CPU vs CPU

Após selecionar Player vs Player é apenas necessário selecionar as posições dos jokers e jogar normalmente. No caso de selecionar Player vs CPU, é preciso seleccionar o lado do CPU, a sua dificuldade, e as posições dos jokers. Finalmente, no caso de seleccionar CPU vs CPU, é preciso selecionar as dificuldades de cada CPU individualmente e as posições dos jokers.

Descrição:

O Mapello é uma variante do Reversi no qual é adicionado algumas peças extras em comparação com o original, sendo elas as Walls (peças que servem como as paredes do tabuleiro e não podem ser colocadas peças em cima), Bonuses (peças que dão mais pontos a quem os captura) e Jokers (peças normais que pertencem ao jogador atual). O objetivo do jogo é terminar o jogo com o maior número de pontos (pontos = peças + pontos bonus) isto faz-se capturando peças inimigas, tornando as da cor do jogador. Ganha o jogador que tiver mais pontos acumulados no final.

Regras

- Apenas pode colocar uma peça por turno
- No início do jogo, o tabuleiro começa com 4 peças no centro (Ex.: B W | W B), 8 jokers nas bordas do tabuleiro e 8 bonus e 8 paredes no meio do tabuleiro.
- Só pode colocar a peça se consegue capturar peças inimigas (Ex.:B W W _)
- A peça tem de ser colocada dentro do quadrado interior 8x8.
- Se não conseguir colocar nenhuma peça passa a jogada para o próximo jogador.
- Se os 2 jogadores passarem consecutivos o jogo acaba e ganha quem tiver mais pontos.
- O jogo acaba quando não houver mais jogadas possíveis (Ex.: Tabuleiro completamente cheio).

Links

-https://nestorgames.com/#mapello_detail -https://cardgames.io/reversi/

Representação de Interna do estado do Jogo

O estado do tabuleiro está guardado numa lista de listas (10 x 10) em qual cada posição vai guardar a peça que está aí. Em cada chamada para fazer a jogada passamos o número de bónus colecionados durante o jogo para cada jogador e adiciona esse bónus ao número de peças da mesma cor que cada jogador tem no tabuleiro.

Peças

As peças na lista de listas são representadas da seguinte forma:

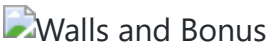
- white : peças brancas
- black : peças pretas
- none : espaço vazio
- wall : parede
- joker : joker
- bonus : bonus

Estado Inicial

No estado inicial temos um tabuleiro N * N gerado pela função na qual preenchemos todos os espaços da lista com 'none' , que em seguida vamos inicializar as posições das peças iniciais (2 peças brancas, 2 peças pretas, 8 paredes, 8 jokers, 8 bonus).

```
% play
% Starts a new game
play:-
    nl, write('###############'), nl,
    write('##  Mapello  ##'), nl,
    write('###############'), nl, nl,
    write('1 - Player vs Player'), nl,
    write('2 - Player vs CPU'), nl,
    write('3 - CPU vs CPU'), nl, nl,
    write('Select game mode: '),
    readOption(Input),
    ((compare(=, Input, 1), nl, setupPvP);
    (compare(=, Input, 2), nl, setupPvC);
    (compare(=, Input, 3), nl, setupCvC);
    (nl, write('Please input 1, 2 or 3, to select the game mode'), nl, play)).
```

Setting Random Walls and Bonus

 Walls and Bonus

Setting Jokers

 Jokers

Estado Intermédio

O estado intermédio do jogo é quando ambos os jogadores conseguem fazer jogadas ou pelo menos um deles consegue jogar, nesta fase vamos adicionando peças brancas e pretas que por sua vez capturam peças dos inimigos tornado-as da mesma cor. No caso de ser colocada uma peça no bonus o jogar recebe 5 pontos extra para o seu score permanente.

Play Loop

 Play

Estado Final

O jogo termina quando uma das 2 condições é atingida:

- Não existem mais espaços vazios no tabuleiro.
- Ambos os jogadores passam as sua vez.

 Game Over

Visualização dos estados do jogo

A visualização do tabuleiro é feita chamando a função:

```
% display_game(+Board, +Player)
% Displays the current game state, and announces next player turn

display_game(Board, Player):-
    printBoard(Board),
    write('Player '),
    write(Player),
    write(' Turn'), nl.
```

Esta função por sua vez chama a função printBoard que por sua vez vai linha a linha e depois elemento a elemento imprimir o conteúdo do tabuleiro.

```
% printBoard(+Board, )
% Displays the current board state
printBoard(Board):-
    nl,
    printHeader,
    printRows(Board, 10), nl.

% printRows(+Board, +N)
% Displays all the rows on the given board
printRows(_, 0).
printRows([FirstElem|OtherElem], N):-
    Number is 10 - N,
    printRowNumber(Number),
    N1 is N-1,
    printRow(FirstElem),
    printRowSep,
    printRows(OtherElem,N1).

% printRow(+Board)
% Displays a single row of the given board
printRow([]):-
    nl.
printRow([FirstElem|OtherElem]):-
    getRep(FirstElem,Label),
    printRep(Label),
    printColumnSep,
    printRow(OtherElem).
```

Cada peça é representada no tabuleiro da seguinte forma:

- 'W' : peças brancas
- 'B' : peças pretas
- '.' : espaço vazio

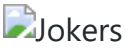
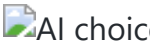
- '#' : parede
- 'J' : joker
- '+' : bonus

 Display

Interações do Utilizador

O utilizador nos vários momentos de utilização são lhe pedidos vários inputs. Os inputs feitos infelizmente não são controlados de qualquer forma, logo inputs incorretos poderão causar problemas na execução do programa.

As várias interações que os utilizadores podem realizar são:

- Escolher o modo jogo (selecionar entre 1 a 3).  Game Mode
- Escolher as posições do Jokers (selecionar 8 coordenadas em que o X e o Y estão entre 0 e 9).  Jokers
- Escolher a posição da peça a jogar (1 coordenada em que o X e o Y estão entre 0 e 9).  Play piece
- Escolher a dificuldade do AI.  AI choice

Lista de Jogadas Válidas

A lista de todas as jogadas válidas é calculada usando a função valid_moves(+BoardState, +Player, -ListOfMoves) que por sua vez retorna todas as posições das jogadas válidas ou um array vazio.

```
% valid_moves(+GameState, +Player, -ListOfMoves)
% Returns a list with all valid moves by the player on the given game state.
valid_moves(GameState, Player, ListOfMoves):-
    checkAllValidMoves(GameState, Player, ListOfMoves, 0, 0).
```

Para verificar quais a jogadas possíveis o predicado chama o predicado checkAllValidMoves(+Board, +Player, -Table, +Y, +X) (que retorna se o Player consegue jogar na posição X,Y) para todas as posições vazias do tabuleiro do jogo.

```
% checkAllValidMoves(+Board, +Player, -Table, +Y, +X)
% returns an array with all possible plays
checkAllValidMoves(_,_,_,10,_).
checkAllValidMoves(Board, Player, [H|T], Y, X):-
    (validatePlay(Board,X,Y,Player) -> H = [X,Y],MOVE is 0 ; MOVE is 1),
    (compare(=,X,9) -> X1 is 0, Y1 is Y + 1; X1 is X + 1, Y1 is Y),
    (compare(=,MOVE,0) -> checkAllValidMoves(Board, Player, T, Y1, X1) ; (compare(=,Y1,10) ->
```

A validação das jogadas é feita usando a função validatePlay(+Board,+X,+Y,+Player) que por sua vez verifica em todas as direções se existe outra peça (joker ou peça da mesma cor) de forma a que consiga capturar pelo menos uma peça inimiga.

```
% validatePlay(+Board,+X,+Y,+Player)
% checks if play made by the player is valid
validatePlay(Board,X,Y,Player):-
    checkInput('Disc', X, Y),
    getPiece(Y,X,Board,Piece),
    \+compare(=, Piece, black),
    \+compare(=, Piece, white),
    \+compare(=, Piece, wall),
    X1 is X - 1,
    X2 is X + 1,
    Y1 is Y - 1,
    Y2 is Y + 1,
    ((checkLeft(Board,X1,Y,Player,0,Pieces1), Pieces1 > 0);
    (checkRight(Board,X2,Y,Player,0,Pieces2), Pieces2 > 0);
    (checkUp(Board,X,Y1,Player,0,Pieces3), Pieces3 > 0);
    (checkDown(Board,X,Y2,Player,0,Pieces4), Pieces4 > 0);
    (checkLeftUp(Board,X1,Y1,Player,0,Pieces5), Pieces5 > 0);
    (checkLeftDown(Board,X1,Y2,Player,0,Pieces6), Pieces6 > 0);
    (checkRightUp(Board,X2,Y1,Player,0,Pieces7), Pieces7 > 0);
    (checkRightDown(Board,X2,Y2,Player,0,Pieces8), Pieces8 > 0)).
```

Execução de Jogadas

A execução de jogadas tem várias funções dependendo do modo de jogo. No caso de Player vs Player é chamado o predicado move(+GameState, +Player, -NewBoard, -Skipped).

```
% move(+GameState, +Player, -NewBoard, -Skipped)
% Goes through a player's turn on the game
move(GameState, 0, NewBoard, Skipped):-
    (canPlay(GameState, black, _), Skipped is 0, placeDiscPlayer1(GameState, NewBoard));
    (copyBoard(GameState, NewBoard), Skipped is 1).

move(GameState, 1, NewBoard, Skipped):-
    (canPlay(GameState, white, _), placeDiscPlayer2(GameState, NewBoard));
    (copyBoard(GameState, NewBoard), Skipped is 1).
```

No caso de Player vs CPU ou CPU vs CPU é chamada o predicado choose_move(+GameState, +Player, +Level, -NewBoard, -Skipped).

```
% choose_move(+GameState, +Player, +Level, -NewBoard, -Skipped)
% Goes through a CPU's turn on the game
choose_move(GameState, 0, Level, NewBoard, Skipped):-
    sleep(2), % purely for aesthetic reasons while watching the CPU's play
    (canPlay(GameState, black, _), Skipped is 0, placeDiscCPU1(GameState, Level, NewBoard));
```

```
    (copyBoard(GameState, NewBoard), Skipped is 1).

choose_move(GameState, 1, Level, NewBoard, Skipped):-
    sleep(2), % purely for aesthetic reasons while watching the CPU's play
    (canPlay(GameState, white, _), Skipped is 0, placeDiscCPU2(GameState, Level, NewBoard));
    (copyBoard(GameState, NewBoard), Skipped is 1).
```

Final do Jogo

A verificação do estado final do jogo é feita usando a função game_over(+Board-,Winner,+Skips), que por sua vez retorna os jogador com o maior score.

```
% game_over(+Board,+Skips,-Winner)
% Returns winner of the game
game_over(Board, Skips, Winner):-
    isGameOver(Board,0,0,Skips),
    nl, write('Game Over!'), nl,
    value(Board,black,V1),
    value(Board,white,V2),
    write('Player 1 (Black) Score: '), write(V1), nl,
    write('Player 2 (White) Score: '), write(V2), nl,
    ((V1 > V2, Winner = 'Player 1 wins!');
    (V2 > V1, Winner = 'Player 2 wins!');
    (V1 = V2, Winner = 'It\'s a draw!')).
```

Avaliação do Tabuleiro

A avaliação do tabuleiro é feita usando o número de peças do player presentes, isto é quanto mais peças presentes no tabuleiro melhor será o score. Esta avaliação é feita usando as value(+GameState,+Player,-Value).

```
% value(+GameState,+Player,-Value)
% Calculates the score of the given player, according to the given game state
value(GameState,black,Value):-
    getAllBlackRowsScores(GameState,0,0,Scores),
    sumlist(Scores,Value).

value(GameState,white,Value):-
    getAllWhiteRowsScores(GameState,0,0,Scores),
    sumlist(Scores,Value).
```

Jogada do Computador

O computador para fazer a sua jogada usa a função choose_move(+GameState, +Player, +Level, -NewBoard, -Skipped) na qual tem comportamentos diferentes dependendo da dificuldade escolhida.

```
% choose_move(+GameState, +Player, +Level, -NewBoard, -Skipped)
% Goes through a CPU's turn on the game
choose_move(GameState, 0, Level, NewBoard, Skipped):-
    sleep(2), % purely for aesthetic reasons while watching the CPU's play
    (canPlay(GameState, black, _), Skipped is 0, placeDiscCPU1(GameState, Level, NewBoard));
    (copyBoard(GameState, NewBoard), Skipped is 1).

choose_move(GameState, 1, Level, NewBoard, Skipped):-
    sleep(2), % purely for aesthetic reasons while watching the CPU's play
    (canPlay(GameState, white, _), Skipped is 0, placeDiscCPU2(GameState, Level, NewBoard));
    (copyBoard(GameState, NewBoard), Skipped is 1).
```

No caso de dificuldade ser fácil, o AI escolhe de todas as jogadas possíveis uma aleatória usando as funções placeDiscCPU1(+Board, +Difficulty, -NewBoard) e placeDiscCPU2(+Board, +Difficulty, -NewBoard).

```
% placeDiscCPU1(+Board, +Difficulty, -NewBoard)
% Places a black Disc (owned by a CPU player 1) on the board
placeDiscCPU1(Board, 1, NewBoard):-
    canPlay(Board, black, ValidMoves),
    length(ValidMoves, MovesAmount),
    random(1, MovesAmount, MoveNumber),
    getMove(ValidMoves, MoveNumber, Move),
    Move = [X,Y],
    setPiece(Board, X, Y, black, TempBoard),
    capturePieces(TempBoard, black, X, Y, NewBoard).

placeDiscCPU1(Board, 2, NewBoard):-
    canPlay(Board, black, ValidMoves),
    getBestMove(Board, ValidMoves, black, 0, _, BestMove),
    BestMove = [X,Y],
    setPiece(Board, X, Y, black, TempBoard),
    capturePieces(TempBoard, black, X, Y, NewBoard).

% placeDiscCPU2(+Board, +Difficulty, -NewBoard)
% Places a white Disc (owned by a CPU player 2) on the board
placeDiscCPU2(Board, 1, NewBoard):-
    canPlay(Board, white, ValidMoves),
    length(ValidMoves, MovesAmount),
    random(1, MovesAmount, MoveNumber),
    getMove(ValidMoves, MoveNumber, Move),
    Move = [X,Y],
    setPiece(Board, X, Y, white, TempBoard),
    capturePieces(TempBoard, white, X, Y, NewBoard).

placeDiscCPU2(Board, 2, NewBoard):-
    canPlay(Board, white, ValidMoves),
    getBestMove(Board, ValidMoves, white, 0, _, BestMove),
    BestMove = [X,Y],
    setPiece(Board, X, Y, white, TempBoard),
    capturePieces(TempBoard, white, X, Y, NewBoard).
```

As jogadas aleatórias são obtidas usando a usando a funçao canPlay(+Board, +Player, -Points) que retorna uma lista de coordenadas de todas as jogadas que possam ser feitas.

```
% canPlay(+Board, +Player, -Points)
% checks if the player can make any plays
canPlay(Board,Player,Points):-
    valid_moves(Board, Player, Points),
    length(Points,N),
    N1 is N - 1, !,
    N1 =\= 0.
```

Conclusões:

Devido a limitações de tempo e alguns erros um bocado limitantes não fomos capazes de implementar tudo de forma funcional e sem erros. Alguns dos erros que foram encontrados ao longo do desenvolvimento do projeto:

- Uma situação semelhante ocorre com a inicialização das peças de bónus e as paredes que ao selecionarem uma posição aleatória se a posição não estiver vazia inserem na mesma essa peça.

Estes erros seriam os importantes de corrigir uma vez que impedem o funcionamento normal do jogo. Também implementar alguma maneira de controlar o input dos utilizadores para impedir erros causados pelos mesmos (Ex.:O utilizador insere uma letra em vez de um número a escolher uma coordenada de uma peça o que provoca um erro e termina). Infelizmente os bónus não conseguiram ser implementados devido a erros que fomos capazes de corrigir, por isso a pontuação é realizada contando o número de peças que cada jogador tem no tabuleiro. É de adicionar que o AI está demasiado simplista neste momento e que talvez se conseguissemos aumentar a sua complexidade através de um algoritmo Alfa-Beta Cut com várias heurísticas para avaliação do tabuleiro tornaria os jogos de CPU vs CPU e Player vs CPU mais interessantes e mais desafiantes.

Bibliografia

- Slides das teóricas
- Manual do SICStus
- https://nestorgames.com/#mapello_detail
- https://cardgames.io/reversi/