Duplo Hex

## Relatório Intercalar

## feup-logo.png

Mestrado Integrado em Engenharia Informatica e

Computação

Programação em Lógica

Grupo: DuploHex\_2

João Manuel Estrada Pereira Gouveia - 201303988

João Pedro Bernardes Mendonça - 201304605

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

**Indice**

1 Introdução 2

2 O Jogo Duplo Hex 2

2.1 História: 2

2.2 Objetivo 2

2.3 Equipamento 2

2.4 Preparação 2

2.5 Acções 2

2.6 Fim do Jogo 3

3 Lógica do Jogo 4

3.1 Representação do estado do jogo 4

3.1.1 Representação das peças no tabuleiro 4

3.1.1 Representação do tabuleiro em vários estados 4

3.2 Visualização do tabuleiro 5

3.3 Lista de Jogadas Válidas 5

3.4 Execução das jogadas 5

3.5 Avaliação do Tabuleiro 6

3.6 Final do Jogo 6

3.7 Jogada do computador 7

4 Interface com o Utilizador 8

5 Conclusões 10

6 Bibliografia 11

**1 Introdução**

O principal objetivo deste projeto é desenvolver capacidades da Programação em Lóogica através da implementação de um jogo de tabuleiro. O jogo escolheido pelo grupo foi o Duplo Hex. Neste relatório encontram-se especificadas as componentes do jogo (condições, jogadas,..), as representações da Interface e alguns detalhes sobre predicados determinantes para o bom funcionamento do jogo.

**2 O Jogo Duplo Hex**

**2.1 História:**

**Foi criado por**  [José Manuel Astilleros García-Monge](http://www.boardgamegeek.com/boardgamedesigner/72319/jose-manuel-astilleros-garcia-monge) e por [Néstor Romeral Andrés](http://www.boardgamegeek.com/boardgamedesigner/9393/nestor-romeral-andres), dono da empresa responsável pela publicação do DuploHex [nestorgames](http://www.boardgamegeek.com/boardgamepublisher/10094/nestorgames) .

**2.2 Objetivo**

**O objetivo do jogo é fazer uma corrente de discos ou aneis da sua cor, nos lados correspondentes.**

O jogo termina quando um jogador conseguir fazer ligar ambos os lados paralelos da sua cor com uma corrente de discos ou uma corrente de anéis.

**2.3 Equipamento**

***DuploHex* é um jogo de ligações para 2 jogadores parecido com o *Hex.* Éconstituído por um tabuleiro hexagonal de 7x7, 25 anéis pretos e 25 anéis brancos, 25 discos pretos e 25 discos brancos(os discos são do tamanho da abertura do anel).**

**2.4 Preparação**

O tabuleiro começa vazio. Cada jogador escolhe uma cor (branco ou preto).

O branco começa por colocar uma das peças, disco ou anel em qualquer célula do tabuleiro.

**2.5 Acções**

Em cada turno, cada jogador pode colocar uma nova peça no tabuleiro e mover outra.

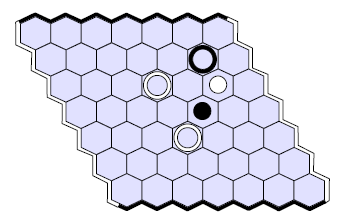
O jogador branco começa por colocar uma das peças, disco ou anel em qualquer célula do tabuleiro. Daqui em diante, começando com o preto, cada jogador terá de fazer uma acção com um disco e uma ação com um anel, a ordem das ações não é importante.

Com um disco é possível fazer **uma e só uma por turno** destas ações:

* Colocar um disco numa célula vazia;
* Mover um disco já existente no tabuleiro para dentro de um anel (branco ou preto) numa célula vizinha. Um disco dentro de um anel não pode ser movido até ao final do jogo.

Com um anel é possível fazer **uma e só uma por turno** destas ações:

Colocar um anel numa célula vazia;

* Mover um anel já existente no tabuleiro para envolver um disco (branco ou preto) existente numa célula vizinha. Um anel com um disco dentro não pode ser movido até ao fim do jogo.

**2.6 Fim do Jogo**

O jogo termina quando um dos jogadores conseguir completar uma corrente de aneis ou discos de um lado para o outro.



Imagem de um turno do jogador preto. Move o anel para envolver o disco branco e move o disco preto para preencher o anel branco

## **3 Lógica do Jogo**

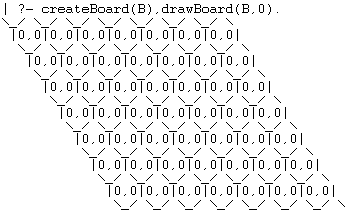
**3.1** **Representação do estado do jogo**

O estado do jogo é guardado recorrendo à base de dados do PROLOG. O tabuleiro de jogo é guardada numa lista de listas, a que cada Lista corresponde a uma célula do tabuleiro . Cada célula poderá conter até 1 disco e 1 anel. Cada célula é guardada na forma [Anel,Disco].

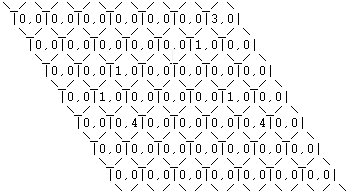
**3.1.1 Representação das peças no tabuleiro**

* **Espaço vazio** - “0”
* **Anel Branco** - “1”
* **Disco Branco** - ”2”
* **Anel** **Preto** - “3”
* **Disco** **Preto** - “4”

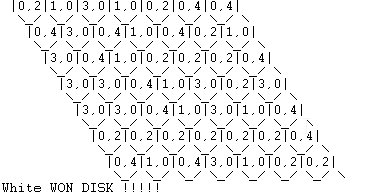
**3.1.1 Representação do tabuleiro em vários estados**



(O tabuleiro encontra-se completamente vazio)



(Estado do tabuleiro passadas algumas jogadas)



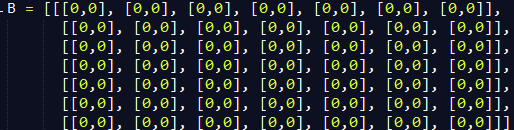
(Possível estado final do tabuleiro)

**3.2 Visualização do tabuleiro**

O tabuleiro de DuploHex é estático (7x7), e é gerado no início do jogo acedendo á função *createBoard..* Durante o jogo, o tabuleiro é mostrado acedendo à função *drawBoard.*

O tabuleiro é passado como argumento de predicado para predicado durante a duração do jogo.

A estrutura onde é guardado o tabuleiro é a seguinte:



(estrutura do tabuleiro)

**3.3 Lista de Jogadas Válidas**

De forma a obter as jogadas válidas, recorre-se ao predicado ***validPlay(Board,Player,X,Y,Xf,Yf,Mode,Pawn)***. ***Board*** é referente ao tabuleiro atual, ***Player*** ao jogador atual, ***X*** ao X inicial, ***Y*** ao Y inicial, ***Xf*** ao X final, ***Yf*** ao Y final, ***Mode*** a se estamos a colocar ou a mover uma peça e ***Pawn*** se estamos a operar um dico ou um anel.

Este predicado verifica se a jogada que estamos a tentar fazer é válida segundo as regras do jogo. Em muitos predicados é utilizado o predicado ***findall*** com a ***validPlay*** a fim de retornar a lista de jogadas possíveis a fazer.

**3.4 Execução das jogadas**

Em cada jogada é colocada uma peça e movida outra, recorrendo aos predicados

Utilizador:

* ***placePawn(Board,Player,Pawn,X,Y,NewBoard)***
* ***movePawn(Board,Player,X,Y,Xf,Yf,Pawn,NewBoard2)***

Bot:

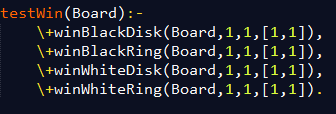
* ***botPlace(Board,Player,NewBoard)***
* ***botMove(Board,Player,NewBoard)***

Cada vez que o Utilizador joga, é verificada a validade da jogada e caso falhe, o backtracking do PROLOG fará com que seja possível fazer novamente a jogada.

No caso do Bot, a validade da jogada é verificada logo à partida, logo esta não falha.

**3.5 Avaliação do Tabuleiro**

Sendo o objetivo do jogo fazer uma corrente de aneis ou discos de um lado ao outro do tabuleiro, é utilizado um predicado para verificar a condição de vitória



(preficado testWin)

que chama outros 4 predicados, cada um verificando se se atingiu a condição de vitória atrevés de algum tipo de peça.

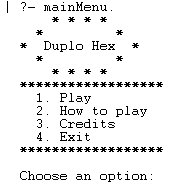
**3.6 Final do Jogo**

Caso a função anterior retorne ***false,*** é anunciado o fim do jogo e é feito um print com o nome do jogador vencedor, Branco ou Preto.

**3.7 Jogada do computador**

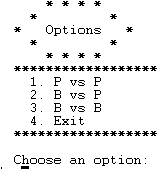
As jogadas do computador são feitas de modo aleatório, verificando todas as jogadas possíveis e sorteando uma delas para ser feita. As jogadas do Bot são executadas através do predicado ***botPlaying(Board,Player,NewBoard).***

## **4 Interface com o Utilizador**

Para se iniciar o jogo, utiliza-se o predicado ***mainMenu,*** que tem o seguinte output: 

(User Interface - mainMenu)

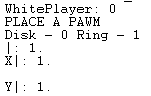
Se o Utilizador escolher a opção ***Play***  é-lhe apresentado outro menu chamado ***gameOptions.***



(User Interface - gameOptions)

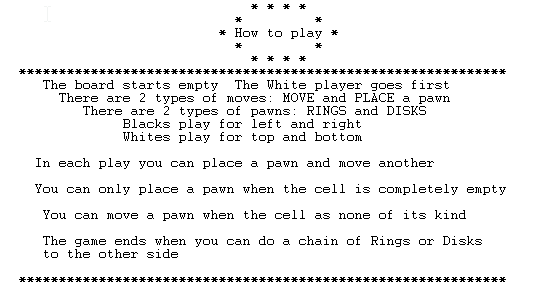
Se o jogador escolher (***P vs P***) é iniciado o jogo de Humano contra Humano, chamando o predicado ***playPvP(Board, Player).***  Escolhendo (***B vs P***) é iniciado jogo de Computador contra Humano, chamando o predicado ***playBvP(Board, Player).***  Escolhendo (***B vs B***) é iniciado jogo de Computador contra Humano, chamando o predicado ***playBvB(Board, Player).*** Escolhendo Exit, o jogo retorna ao ***mainMenu.***

Durante a execução do modo de Utilizador, a interface para a receção das coordenadas das peças a mover é a seguinte:

******

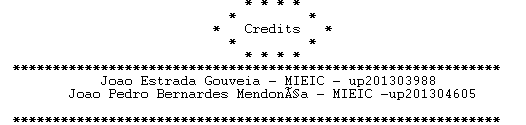
(User Interface – Controlos do Utilizador)

De volta ao mainMenu, escolhendo a opção (***How*** ***to*** ***Play***) é feito um print com a explicação das regras do jogo



(User Interface – How to Play)

Escolhendo a opção ***Credits,*** é feito um print com os membros do grupo que realizaram este projeto.



(User Interface –Credits)

Escolhendo a opção ***Exit***, o jogo termina a sua execução.

## **5 Conclusões**

Este jogo foi de demorada implementação, não só pelas condições de colocações das peças mas pela condição de vitória que levou bastante tempo a ser pensada.

Contudo foi graças a este esforço que nos foi possível estender o nosso conhecimento em reção aos paradigmas de PROLOG e raciocínio lógico.

## **6 Bibliografia**

***http://www.swi-prolog.org/***

## **7 Código fonte**

**:- use\_module(library(lists)).**

**:- use\_module(library(between)).**

**:- use\_module(library(random)).**

**/\* data structure: list 7\*7 [[ring, disk],[ring,disk],...] \*/**

**createBoard(B):-**

**B = [[[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], [0,0], [0,0]],**

**[[0,0], [0,0], [0,0], [0,0], [0,0], [0,0], [0,0]]].**

**/\* Drawing of board \*/**

**drawBlank(0).**

**drawBlank(N):- write(' '),write(' '),N1 is N - 1, drawBlank(N1).**

**drawLine([]):-write('|').**

**drawLine([[Ring,Disk]|Rest]):-write('|'),write(Ring),write(','),write(Disk),drawLine(Rest).**

**drawTopLine(0):-write('/ \\').**

**drawTopLine(N):-write('/ \\ '), N1 is N - 1, drawTopLine(N1).**

**drawBotLine(0):-write('\\').**

**drawBotLine(N):-write('\\\_/ '), N1 is N - 1, drawBotLine(N1).**

**drawBoard([],N):-drawBlank(N),drawBotLine(7),nl.**

**drawBoard([Line|RestBoard],N):-drawBlank(N),drawBotLine(7),nl,**

**write(' '),drawBlank(N),drawLine(Line),**

**nl,**

**N1 is N + 1,**

**drawBoard(RestBoard,N1),!.**

**printRings([]):-**

**printVerticalLine(1),**

**nl.**

**printRings([Elem|Line]):-**

**printVerticalLine(1),**

**emptySpace(1),**

**getRing(Elem,Ring),**

**write(Ring),**

**emptySpace(1),**

**printRings(Line).**

**printDisks([]):-**

**printVerticalLine(1),**

**nl.**

**printDisks([Elem|Line]):-**

**printVerticalLine(1),**

**emptySpace(1),**

**getDisk(Elem,Ring),**

**write(Ring),**

**emptySpace(1),**

**printDisks(Line).**

**displayBoard([]):-**

**printHorizontalLine(29).**

**displayBoard([Line|B]):-**

**displayLine(Line),**

**displayBoard(B).**

**displayLine(Line):-**

**printHorizontalLine(29),**

**printRings(Line),**

**printDisks(Line).**

**printHorizontalLine(0):-**

**nl.**

**printHorizontalLine(NumberOfDashes) :-**

**write('-'),**

**N is NumberOfDashes - 1,**

**printHorizontalLine(N).**

**printVerticalLine(0).**

**printVerticalLine(Number):-**

**write('|'),**

**N is Number - 1,**

**printVerticalLine(N).**

**emptySpace(0).**

**emptySpace(NumberofSpaces):-**

**write(' '),**

**N1 is NumberofSpaces - 1,**

**emptySpace(N1).**

**placeWhiteRing(\_):-**

**write('1').**

**placeWhiteDisk(\_):-**

**write('2').**

**placeBlackRing(\_):-**

**write('3').**

**placeBlackDisk(\_):-**

**write('4').**

**/\* Movements \*/**

**/\* Auxilary Predicates \*/**

**%Mode : 0 - Putting a piece**

**%Mode : 1 - Moving a piece**

**%Pawn : 0 - disk**

**%Pawn : 1 - ring**

**%Atribui um valor ao disco indicado por X e Y**

**getRing([Ring,\_],Ring).**

**getDisk([\_,Disk],Disk).**

**setDisk(Board,X,Y,Disk,NewBoard):- getRing(Board,X,Y,Ring),setMatrix(Board,X,Y,[Ring,Disk],NewBoard).**

**setRing(Board,X,Y,Ring,NewBoard):- getDisk(Board,X,Y,Disk), setMatrix(Board,X,Y,[Ring,Disk],NewBoard).**

**setMatrix(Matrix,X,Y,Elem,NewMatrix):-getListFromMatrix(Matrix,Y,List), setList(List,X,Elem,NewList), setList(Matrix,Y,NewList,NewMatrix).**

**getListFromMatrix(Matrix,N,List):-nth1(N,Matrix,List).**

**setList([\_|Rest],1,NewElem,[NewElem|Rest]):-!.**

**setList([H|Rest],N,NewElem,[H|NewRest]):- N1 is N - 1,setList(Rest,N1,NewElem,NewRest).**

**getElem(Board,X,Y,Elem):-nth1(Y,Board,Line),nth1(X,Line,Elem).**

**getRing(Board,X,Y,Ring):-getElem(Board,X,Y,[Ring,\_]).**

**getDisk(Board,X,Y,Disk):-getElem(Board,X,Y,[\_,Disk]).**

**verifyEmpty([Ring,Disk]):-(Ring =:= 0, Disk =:= 0).**

**checkEmptyList([]):-fail,!.**

**checkEmptyList(\_):-!.**

**% verifies the adjacent cells**

**verifyAdjacent(X,Y,Xf,Yf):- Y1 is Y - 1,Y2 is Y + 1, between(Y1,Y2,YIndex),YIndex>0,YIndex<8,Yf is YIndex,(**

**(YIndex =:= Y1,XF is X + 1,(Xf is X;(XF < 8,Xf is XF)));**

**(YIndex =:= Y,XF1 is X - 1, XF2 is X + 1,((XF1 > 0, Xf is XF1);(XF1 <8,Xf is XF2)));**

**(YIndex =:= Y2,XF is X - 1,(Xf is X;(XF > 0,Xf is XF)))**

**).**

**%validates the plays that the user and the bot make**

**validPlay(Board,Player,X,Y,Xf,Yf,Mode,Pawn):-**

**between(1,7,Xf),between(1,7,Yf),between(1,7,X),between(1,7,Y),**

**getElem(Board,Xf,Yf,Elem),**

**((Mode =:= 0,**

**verifyEmpty(Elem));**

**(Mode =:= 1,**

**verifyAdjacent(X,Y,Xf,Yf),**

**((Pawn =:= 1, Player =:= 0, getRing(Board,X,Y,Ring), getDisk(Board,X,Y,Disk),getRing(Board,Xf,Yf,NewRing),Disk =:= 0,Ring =:= 1, NewRing =:= 0);**

**(Pawn =:= 0, Player =:= 0, getDisk(Board,X,Y,Disk),getRing(Board,X,Y,Ring),getDisk(Board,Xf,Yf,NewDisk),Ring =:=0,Disk =:= 2, NewDisk =:= 0);**

**(Pawn =:= 1, Player =:= 1, getRing(Board,X,Y,Ring), getDisk(Board,X,Y,Disk),getRing(Board,Xf,Yf,NewRing),Disk =:= 0,Ring =:= 3, NewRing =:= 0);**

**(Pawn =:= 0, Player =:= 1, getDisk(Board,X,Y,Disk),getRing(Board,X,Y,Ring),getDisk(Board,Xf,Yf,NewDisk),Ring=:=0,Disk =:= 4, NewDisk =:= 0)))).**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**%PLAY CYCLES**

**%pvp cycle predicate**

**playPvP(Board, Player):-**

**testWin(Board),**

**drawBoard(Board,0),**

**((Player =:= 0, write('White'), write('Player: '), write(Player),nl,**

**playPlaceAux(Board,Player,NewBoard),**

**drawBoard(NewBoard,0),**

**playMoveAux(NewBoard,Player,NewBoard1),**

**Player1 is 1);**

**(Player =:= 1,**

**write('Black'),write('Player: '), write(Player), nl,**

**playPlaceAux(Board,Player,NewBoard),**

**drawBoard(NewBoard,0),**

**playMoveAux(NewBoard,Player,NewBoard1),**

**Player1 is 0)),**

**!, playPvP(NewBoard1,Player1).**

**playPvP(Board,Player):- write('Invalid Input'),nl,nl,play(Board,Player),!.**

**%bvp cycle predicate**

**playBvP(Board,Player):-**

**testWin(Board),**

**drawBoard(Board,0),**

**botPlaying(Board,Player,NewBoard),**

**playPlaceAux(NewBoard,Player,NewBoard1),**

**drawBoard(NewBoard1,0),**

**playMoveAux(NewBoard1,Player,NewBoard2),**

**NextP is 1-Player,**

**playBvP(NewBoard2,NextP).**

**playBvP(\_,\_):- fail,!.**

**%bvb cycle predicate**

**playBvB(Board,Player):-**

**drawBoard(Board,0),**

**testWin(Board),**

**((Player =:= 0, write('White player'));**

**(Player =:=1, write('Black player'))),**

**botPlaying(Board,Player,NewBoard),**

**NextP is 1-Player,**

**playBvB(NewBoard,NextP).**

**playBvB(\_,\_):- fail,!.**

**playPlaceAux(Board,Player,NewBoard):-**

**placePawnAux(Board,Player,NewBoard),!.**

**playPlaceAux(Board,Player,NewBoard):- write('Invalid Input'),nl,nl,playPlaceAux(Board,Player,NewBoard),!.**

**playMoveAux(Board,Player,NewBoard):-**

**movePawnAux(Board,Player,NewBoard),!.**

**playMoveAux(Board,Player,NewBoard):- write('Invalid Input'),nl,nl,playMoveAux(Board,Player,NewBoard),!.**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**%BOT PREDICATES**

**botPlaying(Board,Player,NewBoard):-**

**write('Player: '), write(Player), nl,nl,**

**botPlace(Board,Player,NewBoard),**

**drawBoard(NewBoard,0),**

**botMove(NewBoard,Player,\_).**

**%bot placing pawns**

**botPlace(Board,Player,NewBoard):-**

**random(0,2,Pawn),**

**findall([Xf,Yf],validPlay(Board,Player,1,1,Xf,Yf,0,Pawn),L),**

**checkEmptyList(L),**

**length(L,Size), random(0,Size,Index), nth0(Index,L,Cell),**

**nth0(0,Cell,X), nth0(1,Cell,Y),**

**placePawn(Board,Player,Pawn,X,Y,NewBoard).**

**botPlace(\_,\_,\_):- write('Cannot place any pawn'),nl,!.**

**%bot moving pawns**

**botMove(Board,Player,NewBoard):-**

**random(0,2,Pawn),**

**findall([X,Y,Xf,Yf],validPlay(Board,Player,X,Y,Xf,Yf,1,Pawn),L),**

**checkEmptyList(L),**

**length(L,Size), random(1,Size,Index), nth0(Index,L,Cell),**

**nth0(0,Cell,X1),nth0(1,Cell,Y1),nth0(2,Cell,X1f),nth0(3,Cell,Y1f),**

**movePawn(Board,Player,X1,Y1,X1f,Y1f,Pawn,NewBoard).**

**botMove(\_,\_,\_):-write('Cannot move any pawn'),nl,!.**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**%USER PREDICATES**

**placePawnAux(Board, Player, NewBoard):-**

**write('PLACE A PAWM'),nl,**

**write('Disk - 0 Ring - 1'),nl, read(Ans),**

**write('X'), read(Xf),nl,**

**write('Y'), read(Yf),nl,**

**placePawn(Board,Player,Ans,Xf,Yf,NewBoard).**

**%user placing pawns**

**placePawn(Board,Player,Pawn,X,Y,NewBoard):-**

**validPlay(Board, \_, \_, \_,X,Y,0, \_),**

**((Pawn =:= 0, ((Player =:= 0, Disk is 2);**

**(Player =:= 1, Disk is 4)),**

**setDisk(Board,X,Y,Disk,NewBoard));**

**(Pawn =:= 1, ((Player =:= 0, Ring is 1);**

**(Player =:= 1, Ring is 3)),**

**setRing(Board,X,Y,Ring,NewBoard))).**

**placePawn(Board,Player,\_,\_,\_,NewBoard):- write('Invalid Play'),nl,nl, placePawnAux(Board,Player,NewBoard),!.**

**movePawnAux(Board,Player,NewBoard):-**

**write('MOVE A PAWN'),nl,**

**write('X'), read(X),nl,**

**write('Y'), read(Y),nl,**

**getRing(Board,X,Y,Ring), getDisk(Board,X,Y,Disk),**

**write('RING: '), write(Ring), write(' Disk: '), write(Disk),nl,**

**write('Disk - 0 Ring - 1'),nl, read(Ans),**

**((Ans =:= 0, Player =:= 0, Disk =\= 2, write('ERROR'),nl,Test is 0);**

**(Ans =:= 0, Player =:= 1, Disk =\= 4,write('ERROR'),nl,Test is 0);**

**(Ans =:= 1, Player =:= 0, Ring =\= 1,write('ERROR'),nl,Test is 0);**

**(Ans =:= 1, Player =:= 1, Ring =\= 3,write('ERROR'),nl,Test is 0);**

**(Ans =:= 0, Player =:= 0, Disk =:= 2, Pawn is 0, write('Move to:'),nl,write('Xf'), read(Xf),nl, write('Yf'), read(Yf),nl,Test is 1);**

**(Ans =:= 0, Player =:= 1, Disk =:= 4, Pawn is 0, write('Move to:'),nl,write('Xf'), read(Xf),nl,write('Yf'), read(Yf),nl,Test is 1);**

**(Ans =:= 1, Player =:= 0, Ring =:= 1, Pawn is 1, write('Move to:'),nl,write('Xf'), read(Xf),nl,write('Yf'), read(Yf),nl,Test is 1);**

**(Ans =:= 1, Player =:= 1, Ring =:= 3, Pawn is 1, write('Move to:'),nl,write('Xf'), read(Xf),nl,write('Yf'), read(Yf),nl,Test is 1)),**

**((Test =:= 0, !,movePawnAux(Board,Player, NewBoard));**

**(Test=:=1, movePawn(Board,Player,X,Y,Xf,Yf,Pawn,NewBoard))).**

**%user moving pawns**

**movePawn(Board,Player,X,Y,Xf,Yf,Pawn,NewBoard2):-**

**validPlay(Board,Player,X,Y,Xf,Yf,1,Pawn),**

**((Player =:= 0,((Pawn =:= 0, setDisk(Board,X,Y,0,NewBoard), setDisk(NewBoard,Xf,Yf,2,NewBoard2));**

**(Pawn =:= 1, setRing(Board,X,Y,0,NewBoard), setRing(NewBoard,Xf,Yf,1,NewBoard2))));**

**(Player =:= 1, ((Pawn =:= 0, setDisk(Board,X,Y,0,NewBoard), setDisk(NewBoard,Xf,Yf,4,NewBoard2));**

**(Pawn =:= 1, setRing(Board,X,Y,0,NewBoard), setRing(NewBoard,Xf,Yf,3,NewBoard2))))).**

**movePawn(Board,Player,\_,\_,\_,\_,\_,NewBoard):- write('Invalid Play'),nl,nl, movePawnAux(Board,Player,NewBoard),!.**

**%WINING CONDITIONS**

**testWin(Board):-**

**\+winBlackDisk(Board,1,1,[1,1]),**

**\+winBlackRing(Board,1,1,[1,1]),**

**\+winWhiteDisk(Board,1,1,[1,1]),**

**\+winWhiteRing(Board,1,1,[1,1]).**

**/\*------------------------------------------------------------\*/**

**%BLACK DISK WINING CONDITION**

**winBlackDisk(\_,\_,7,\_):-write('BLACK WON DISK !!!!!!'),nl,!.**

**winBlackDisk(Board,X,Y,Searched):-**

**X<8,Y<8,**

**getDisk(Board,X,Y,NewDisk),**

**NewDisk = 4,**

**findall([Xf,Yf],verifyAdjacent(X,Y,Xf,Yf),L),**

**verifyBlackDiskExistence(Board,L,Searched).**

**winBlackDisk(\_,8,\_,\_):-fail,!.**

**winBlackDisk(Board,X,Y,Searched):-X<8,NextX is X+1, winBlackDisk(Board,NextX,Y,Searched),!.**

**verifyBlackDiskExistence(\_,[],\_,\_):-!.**

**verifyBlackDiskExistence(Board,[L|\_],Searched):-**

**\+(member(L,Searched)),**

**append(Searched,[L],NewSearched),**

**nth0(0,L,X),nth0(1,L,Y),**

**getDisk(Board,X,Y,NewDisk), NewDisk = 4,**

**winBlackDisk(Board,X,Y,NewSearched),!.**

**verifyBlackDiskExistence(Board,[\_|LTail], Searched):- verifyBlackDiskExistence(Board,LTail,Searched),!.**

**/\*------------------------------------------------------------\*/**

**%BLACK RING WINING CONDITION**

**winBlackRing(\_,\_,7,\_):-write('BLACK WON RING !!!'),nl,!.**

**winBlackRing(Board,X,Y,Searched):-**

**X<8, Y<8,**

**getRing(Board,X,Y,NewRing),**

**NewRing = 3,**

**findall([Xf,Yf],verifyAdjacent(X,Y,Xf,Yf),L),**

**verifyBlackRingExistence(Board,L,Searched).**

**winBlackRing(\_,8,\_,\_):-fail,!.**

**winBlackRing(Board,X,Y,Searched):-X<8, NextX is X+1, winBlackRing(Board,NextX,Y,Searched),!.**

**verifyBlackRingExistence(\_,[],\_,\_):-!.**

**verifyBlackRingExistence(Board,[L|\_],Searched):-**

**\+(member(L,Searched)),**

**append(Searched,[L],NewSearched),**

**nth0(0,L,X),nth0(1,L,Y),**

**getRing(Board,X,Y,NewRing), NewRing = 3,**

**winBlackRing(Board,X,Y,NewSearched),!.**

**verifyBlackRingExistence(Board,[\_|LTail], Searched):- verifyBlackRingExistence(Board,LTail,Searched),!.**

**/\*-----------------------------------------------------------------------------------\*/**

**%White DISK WINING CONDITION**

**winWhiteDisk(\_,7,\_,\_):-write('White WON DISK !!!!!'),nl,!.**

**winWhiteDisk(Board,X,Y,Searched):-**

**X<8, Y<8,**

**getDisk(Board,X,Y,NewDisk),**

**NewDisk = 2,**

**findall([Xf,Yf],verifyAdjacent(X,Y,Xf,Yf),L),**

**verifyWhiteDiskExistence(Board,L,Searched).**

**winWhiteDisk(\_,\_,8,\_):-fail,!.**

**winWhiteDisk(Board,X,Y,Searched):-Y<8,NextY is Y+1, winWhiteDisk(Board,X,NextY,Searched),!.**

**verifyWhiteDiskExistence(\_,[],\_,\_):-!.**

**verifyWhiteDiskExistence(Board,[L|\_],Searched):-**

**\+(member(L,Searched)),**

**append(Searched,[L],NewSearched),**

**nth0(0,L,X),nth0(1,L,Y),**

**getDisk(Board,X,Y,NewDisk), NewDisk = 2,**

**winWhiteDisk(Board,X,Y,NewSearched),!.**

**verifyWhiteDiskExistence(Board,[\_|LTail], Searched):- verifyWhiteDiskExistence(Board,LTail,Searched),!.**

**/\*------------------------------------------------------------------------------\*/**

**%WHITE RING WINING CONDITION**

**winWhiteRing(\_,7,\_,\_):-write('White WON RING !!!!'),nl,!.**

**winWhiteRing(Board,X,Y,Searched):-**

**X<8, Y<8,**

**getRing(Board,X,Y,NewRing),**

**NewRing = 1,**

**findall([Xf,Yf],verifyAdjacent(X,Y,Xf,Yf),L),**

**verifyWhiteRingExistence(Board,L,Searched).**

**winWhiteRing(\_,\_,8,\_):-fail,!.**

**winWhiteRing(Board,X,Y,Searched):-Y<8,NextY is Y+1, winWhiteRing(Board,X,NextY,Searched),!.**

**verifyWhiteRingExistence(\_,[],\_,\_):-!.**

**verifyWhiteRingExistence(Board,[L|\_],Searched):-**

**\+(member(L,Searched)),**

**append(Searched,[L],NewSearched),**

**nth0(0,L,X),nth0(1,L,Y),**

**getRing(Board,X,Y,NewRing), NewRing = 1,**

**winWhiteRing(Board,X,Y,NewSearched),!.**

**verifyWhiteRingExistence(Board,[\_|LTail], Searched):- verifyWhiteRingExistence(Board,LTail,Searched),!.**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**%USER INTERFACE**

**mainMenu:-**

**printMainMenu,**

**read(Input),**

**((Input =:= 1, gameOptions, mainMenu);**

**(Input =:= 2, printHowToPlay, mainMenu);**

**(Input =:= 3, printCredits, mainMenu);**

**(Input =:= 4);**

**(nl, write('Error: invalid input.'),nl, mainMenu)).**

**printMainMenu:-**

**write(' \* \* \* \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* Duplo Hex \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* \* \* \* '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' 1. Play '), nl,**

**write(' 2. How to play '), nl,**

**write(' 3. Credits '), nl,**

**write(' 4. Exit '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' '), nl,**

**write(' Choose an option: '), nl.**

**gameOptions:-**

**printGameOptions,**

**read(Input),**

**createBoard(B),**

**((Input =:= 1, playPvP(B,0));**

**(Input =:= 2, playBvP(B,0));**

**(Input =:= 3, playBvB(B,0));**

**(Input =:= 4);**

**(nl, write('Error: invalid input.'),nl, mainMenu)).**

**printGameOptions:-**

**write(' \* \* \* \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* Options \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* \* \* \* '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' 1. P vs P '), nl,**

**write(' 2. B vs P '), nl,**

**write(' 3. B vs B '), nl,**

**write(' 4. Exit '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' '), nl,**

**write(' Choose an option: '), nl.**

**printHowToPlay:-**

**write(' \* \* \* \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* How to play \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* \* \* \* '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' The board starts empty The White player goes first '), nl,**

**write(' There are 2 types of moves: MOVE and PLACE a pawn '), nl,**

**write(' There are 2 types of pawns: RINGS and DISKS '), nl,**

**write(' Blacks play for left and right '), nl,**

**write(' Whites play for top and bottom '), nl,**

**write(' '), nl,**

**write(' In each play you can place a pawn and move another '), nl,**

**write(' '), nl,**

**write(' You can only place a pawn when the cell is completely empty '), nl,**

**write(' '), nl,**

**write(' You can move a pawn when the cell as none of its kind '), nl,**

**write(' '), nl,**

**write(' The game ends when you can do a chain of Rings or Disks '), nl,**

**write(' to the other side '), nl,**

**write(' '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl.**

**printCredits:-**

**write(' \* \* \* \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* Credits \* '), nl,**

**write(' \* \* '), nl,**

**write(' \* \* \* \* '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl,**

**write(' Joao Estrada Gouveia - MIEIC - up201303988 '), nl,**

**write(' Joao Pedro Bernardes Mendonça - MIEIC -up201304605 '), nl,**

**write(' '), nl,**

**write(' \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '), nl.**