# PENTALATH IN PROLOG

**Relatório Final**


Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo Pentalath_1:**
Inês Teixeira- up201404592
Pedro Costa - up201403291

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

14 de Novembro de 2016

# Resumo

O projeto consiste na implementação do jogo Pentalath em Prolog. Este jogo é um jogo de tabuleiro de dois jogadores que tem como objetivo formar um cinco em linha e capturar peças inimigas.

Todas as regras do jogo foram implementadas, à exceção da captura de peças quando estas ficam rodeadas por peças inimigas. O bot está tambem implementado com todas as opções - PLayer vs. Payer, Player vs. CPU, CPU vs. CPU - e possui 2 níveis de dificuldade.

Este projeto serviu para a consolidação dos nossos conhecimentos em Prolog, desta forma, considerando o desenvolvimento deste jogo muito produtivo.

# Conteúdo

# 1 Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Progamação Lógica do 3º ano do MIEIC.

O objetivo era implementar um jogo em Prolog. Uma das motivações que nos levou a escolher o jogo Pentalath foi o facto de este ter sido criado por um computador e poder aprender com este projeto uma nova linguagem de programação.

# 2 O Jogo PENTALATH

Pentalath é um jogo de tabuleiro de dois jogadores que tem como finalidade isolar o inimigo de forma a capturar as suas peças e formar linhas de cinco peças, ou mais, da mesma cor.

Este jogo, criado pelo computador 'Ludi' em 2007 junta várias regras de jogos de tabuleiro tradicionais como o '5 em Linha' e o 'Capture Go', mas com algumas particularidades como a sua forma, um trapézio com o total de 70 células hexagonais, como é possível ver na figura 1, abaixo representada.

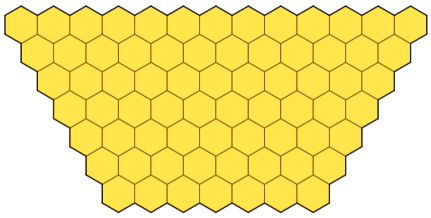

Figura 1: Tabuleiro do jogo

O tabuleiro começa vazio, no entanto, cada jogador tem 40 peças, brancas ou pretas.

Na figura 2,pode-se ver um exemplo de uma jogada onde a peça branca 6 interseta a hipótese de a equipa preta formar uma linha e vencer:

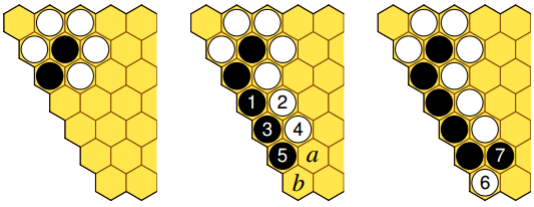


Figura 2: Exemplo de jogada

A cada jogada, se uma peça for rodeada por peças da cor oposta, então a peça é capturada do tabuleiro, como é possível verificar na figura 3. As peças pretas, após a colocação da peça branca com a letra 'f', estão rodeadas pelas peças brancas, dessa forma devem ser retiradas do tabuleiro e não podem ser jogadas.

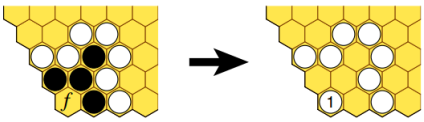

Figura 3: Exemplo de captura de peças inimigas

Desta forma, Pentalath é um jogo de estratégia, onde o jogador deve ter em mente diferentes objetivos- estender uma linha de cinco peças iguais enquanto captura as peças do inimigo.

# 3 Lógica do Jogo

## 3.1 Representação do Estado do Jogo

O tabuleiro é formado por uma lista de listas e é criado de forma dinâmica, dependendo do número de colunas e linhas que seja definido pelo utilizador. Como a forma do tabuleiro é trapezoidal, cada linha tem menos uma casa que a anterior.

Como a forma do tabuleiro é trapezoidal, cada linha tem menos uma casa que a anterior, o modo de tabulereiro *default* apresenta treze colunas para a primeira linha e apenas 7 na ultima linha, com 7 linhas no total.

No entanto, de forma a dinamizar o jogo e a criar vários níveis de dificuldades, o tabuleiro é formado por uma lista de listas e é criado de forma dinâmica, dependendo do número de colunas e linhas que seja definido pelo utilizador.

A lista de listas é criado através dos seguintes predicados representados no listing 1, presente no Anexo.

A cada nova linha é introduzida com menos uma casa que a linha anterior.

Cada uma da casas vazias é representada por 'null'.

A matriz inicial, sem peças, para o tabuleiro de tamanho predefinido nas regras é:

```
board([
[null,null,null,null,null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null]
])
```

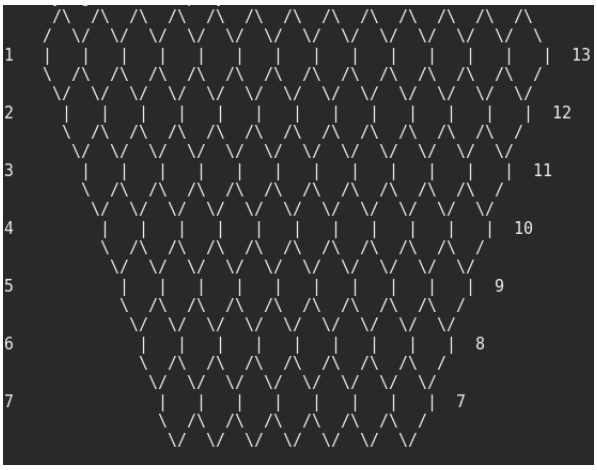

Figura 4: Representação gráfica do tabuleiro vazio.

Exemplo de matriz intermédia, após algumas jogadas:

board([
[**black**,**black**,null,null,null,null,null,null,null,null,null,null,null],
[null,null,**black**,null,null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,**black**,null,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null,null,null],
[**white**,**white**,**white**,**white**,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null]
])

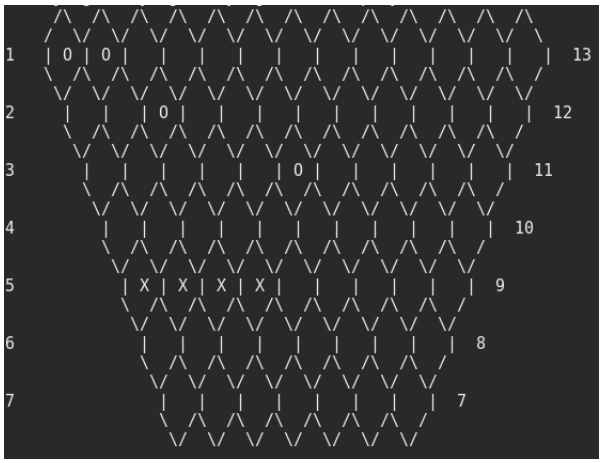

Figura 5: Representação gráfica de um tabuleiro intermedio.

Exemplo de matriz final, aquando numa situação de vitória pelas peças bran-

cas:

board([
[**black**,**black**,null,null,null,null,null,null,null,null,null,null,null],
[null,null,**black**,null,**black**,**black**,**black**,null,null,null,null,null],
[null,null,**black**,null,null,**white**,null,null,null,null,null],
[null,null,null,null,null,null,null,null,null,null,null],
[**white**,**white**,**white**,**white**,**white**,null,null,null,null],
[null,null,null,null,null,null,null,null],
[null,null,null,null,null,null,null]
])

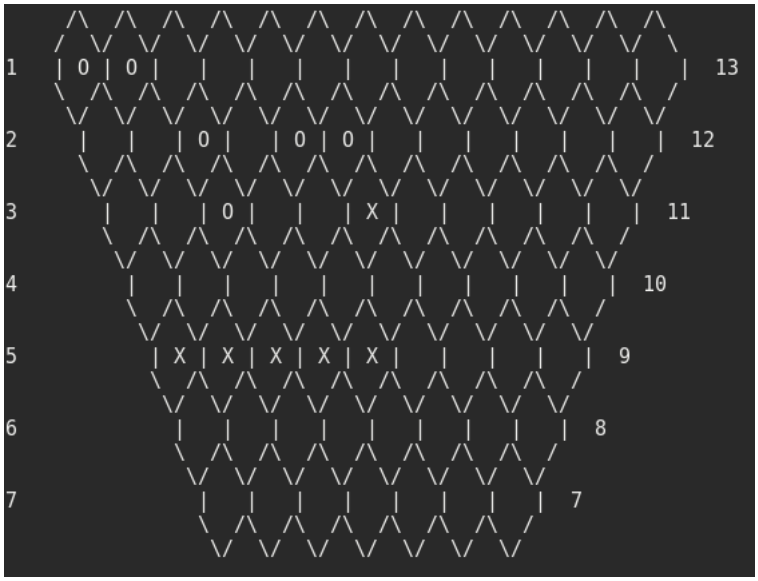


Figura 6: Representação gráfica do tabuleiro final.

## 3.2 Visualização do Tabuleiro

Para representar os diferentes estados das casas do tabuleiro vão ser usados três símbolos diferentes: a letras 'B' (peças brancas), a letra 'P' (peças pretas) e o ' '(espaço, casa vazia).

A função que imprime o tabuleiro é a display_board, representada no listing 2 no Anexo.

Com a seguinte representação gráfica:

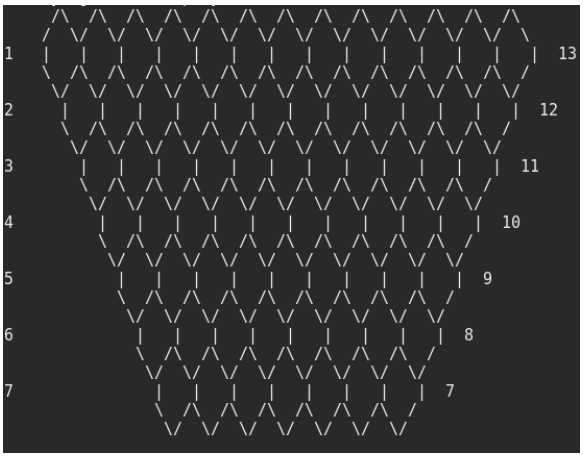

Figura 7: Representação gráfica do tabuleiro

## 3.3 Execução de Jogadas

A jogadas quer realizadas por utilizador ou Bot seguem sempre a mesma função, *playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, 1,DIFFICULTY)* (encontra-se em anexo, no listing 3), primeiro é feito o display do board, de seguida pedidas as coordenadas através da função *getCoordinates(T,X,Y,MAX_X,MAX_Y,PLAYER,MODE,DIFFICULTY)* (encontra-se em anexo, no listing 4) e depois a peça é colocada nas coordenadas pela função *replace(T,Y,X,PLAYER,B)* (encontra-se em anexo, no listing 5) que recebe o tabuleiro, as coordenadas, a peça a colocar e o tabuleiro resultante. De seguida é efetuada a verificação do fim de jogo e o display do board.

No caso especifico do bot, a função getCoordinates para o bot em modo facil gera coordenadas válidas de forma aleatória, e para o caso do modo dificil, ele procura a melhor posição para colocar a próxima peça de forma a fazer 5 em linha, através da função *intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X, MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND)* (encontra-se em anexo, no listing 6).

## 3.4 Final do Jogo

Por forma a verificar o fim do jogo temos 4 funções auxiliares.

A primeira, *verify_full_board*, chamada antes de qualquer jogada, verifica se o tabuleiro se encontra cheio, se sim indica fim do jogo com empate.

As outras 3 funções( *verify_horizontal*, *verify_diagonals_rigth* e *verify_diagonals_left* ), chamadas após a colocação de uma peça, verificam a existência de 5 peças em linha do mesmo jogador, na horizontal e nas diagonais.

## 3.5 Jogada do Computador

O utilizador pode escolher entre dois níveis de dificuldade.

O primeiro nível, o CPU coloca aleatoriamente uma posição no tabuleiro.

No segundo nível de dificuldade, o CPU inicialmente coloca a peça numa casa aleatória. Nas próximas jogadas, se houver uma peça sua no tabuleiro, coloca a nova do lado esquerdo ou direito, tentando, ao longo das jogadas, fazer

um cinco em linha horizontal. Se não houver qualquer peça livre, ou seja, peças brancas a bloquearem, o tabuleiro coloca a peça numa casa aleatória.

Os predicados que serviram para a implementação dos níveis de dificuldade encontram-se no listing X e X, no Anexo, respetivamente.

# 4 Interface com o Utilizador

O jogo inicia com o Menu principal. Este menu apresenta 4 opções.


Figura 8: Main Menu

Quando o utilizador escolhe a primeira opção (Player vs. Player) é iniciado de seguida o jogo, onde o tabuleiro é desenhado e é pedido as coordenadas X e Y aos jogadores. Isto repete-se ao longo de todas as jogadas até algum deles ganhar ou o tabuleiro ficar cheio.

A segunda e terceira opção referem-se às opçoes Player vs. CPU e CPU vs. CPU, respetivamente. É nestas opções que é pedida ao utilizador que indique o nível de dificuldade em que pretende jogar. De seguida, o jogo decorre normalmente. O CPU coloca as suas peças de acordo com o nível de dificuldade escolhido, enquanto que, se tiver sido escolhida a opção 2, é pedida ainda as coordenadas da nova peça ao utilizador.

Na opção quatro, o utilizador entra num menu onde é explicada ao utilizador como jogar o Pentalath, detalhadamente.

# 5 Conclusões

A realização deste projeto serviu para aprofundar os conhecimentos adquiridos ao longo das aulas e no geral, consideramos que fizemos um bom trabalho.

Todos os nosso predicados estão simplificados, uma vez que foi sempre prioritário na nossa implementação o uso de predicados auxiliares de forma a que todo o nosso código fosse de fácil compreensão e baixa complexidade.

O principal aspeto onde o projeto poderia ter sido melhorado era na adição de níveis de dificuldade no bot, com o uso de Inteligência Artificial, no entanto, por falta de tempo, não foi possível adicionar ao jogo uma inteligencia artificial completa onde fosse possível haver vários níveis de dificuldade e nao apenas dois.


Figura 9: Tabuleiro do jogo e Escolha de Difuculdade


Figura 10: Menu Como Jogar

Concluindo, consideramos que grande parte das nossas dificulades foram ultrupassadas, conseguindo construir um jogo sem erros e fluído, onde o bot funciona sem quaisquer problemas e os nossos conhecimentos em Prolog foram ampliados.

# A Anexo do Código Implementado

**Representação do Estado do Jogo:**

Listing 1: **Código para criar dinamicamente o tabuleiro**

```
init_list(R, [ ]) :-
    R =< 0, !.
init_list(R, [_ | T]) :-
    R > 0,
    R2 is R - 1,
    init_list(R2, T).

init_matrix(_, R, []) :-
    R =< 0, !.

%init_matrix(Columns,Rows,Board)
init_matrix(C, R, [H|T]) :-
    init_list(C,H),
    R2 is R - 1,
    init_matrix(C-1, R2, T).
```

**Visualização do Tabuleiro:**

Listing 2: **Display do Tabuleiro**

```
display_walls(S,SI):-
      S > 0,
      SI < 9,
      S1 is S - 1,
      S2 is SI + 1,
      write(''\' '),
      write(' '),
      write('/'),
      display_walls(S1,S2).

    display_walls(S,SI):-
      S > 0,
      S1 is S - 1,
      S2 is SI + 1,
      write(''\''),
      write('  '),
      write('/'),
      display_walls(S1,S2).

    display_walls(S,SI):-nl.

    display_walls2(S):-
      S > 0,
      S1 is S - 1,
      write(' '\''),
```

```
  write('/  '),
  display_walls2(S1).

display_walls2(S):-nl.

display_spaces(SP):-
  SP > 0,
  SP1 is SP - 1,
  write('   '),
  display_spaces(SP1).

display_spaces(SPA):-
  write('').

display_board_top(S):-
  S > 0,
  S1 is S - 1,
  write(' /'),
  write(''\' '),
  display_board_top(S1).

display_board_top(S):-nl.


display_board_top2(S):-
  S > 0,
  S1 is S - 1,
  write('/   '),
  write(''\''),
  display_board_top2(S1).

display_board_top2(S):-nl.


display_board([L1|LS],SIZE,SPACE,TOP,SI):-
  SIZE > 7,
  SIZE1 is SIZE - 1,
  SPACE1 is SPACE + 1,
  TOP > 0,
  display_spaces(SPACE),
  display_board_top(SIZE-1),
  display_spaces(SPACE),
  display_board_top2(SIZE-1),
  INDEX is SPACE1 -  2,
  write(INDEX),
  display_spaces(SPACE-1),
  write(' |'),
  display_line(L1),
  write('   '),
  SIZE2 is SIZE-1,
  write(SIZE2),
  nl,
```

```
    display_spaces(SPACE),
    display_walls(SIZE1,SI),
    display_spaces(SPACE),
    display_walls2(SIZE1),
    display_board(LS,SIZE1,SPACE1,0,SI).

    display_board([L1|LS],SIZE,SPACE,TOP,SI):-
      SIZE > 7,
      SIZE1 is SIZE - 1,
      SPACE1 is SPACE + 1,
      INDEX is SPACE1 -  2,
      write(INDEX),
      display_spaces(SPACE-1),
      write(' |'),
      display_line(L1),
      write('  '),
      SIZE2 is SIZE-1,
      write(SIZE2),
      nl,
      display_spaces(SPACE),
      display_walls(SIZE1,SI),
      display_spaces(SPACE),
      display_walls2(SIZE1),
      display_board(LS,SIZE1,SPACE1,0,SI).

  display_board([],SIZE,SPACE1,TOP,SI):-nl.

  display_line([E1|ES]):-
    traduz(E1,V),
    write(V),
    write('|'),
    display_line(ES).
  display_line([]):-write('').

  traduz(null      ,'    ').
  traduz(b         ,' X  ').
  traduz(p         ,' O  ').
  traduz(E1        ,'').
```

**Execução de Jogadas:**

Listing 3: **pLayGame**

```
playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, 1):-
  verify_full_board(T, 0, 0, MAX_X, MAX_Y, VERIFY, p),
  playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, VERIFY).

playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, 2):-
  display_board(T,14,2,1,0).

playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, VERIFY):-
```

```
  GAME_END == b,
  write('PLAYER '),
  traduz(GAME_END,V),
  write(V),
  write(' WON'),nl.
  display_board(T,14,2,1,0).

playGame(T, MAX_X, MAX_Y, Z, COUNT, MODE, GAME_END, VERIFY):-
  GAME_END == p,
  write('PLAYER '),
  traduz(GAME_END,V),
  write(V),
  write(' WON'),nl.
  display_board(T,14,2,1,0).

playGame(T, MAX_X, MAX_Y, b, COUNT, MODE, GAME_END, VERIFY):-
  write('It is player '),
  traduz(b,V),
  write(V),
  write(' turn.'),nl,
  display_board(T,14,2,1,0),
  getCoordinates(T,X,Y,MAX_X,MAX_Y, b, MODE),
  replace(T,Y,X,b,B),
  verify_horizontal(B,b,X,X,Y,MAX_X,COUNT, GAME_END),
  verify_diagonals_right(B,b,Y,X,Y, MAX_Y,COUNT,GAME_END),
  verify_diagonals_left(B,b,X,Y,X,Y,MAX_X,MAX_Y,COUNT,GAME_END),
  COUNT < 5,
  display_board(B,14,2,1,0),
  playGame(B, MAX_X, MAX_Y, p, COUNT,MODE, GAME_END, 1).


playGame(T, MAX_X, MAX_Y, p, COUNT, MODE, GAME_END, VERIFY):-
  write('It is player '),
  traduz(p,V),
  write(V),
  write(' turn.'),nl,
  display_board(T,14,2,1,0),
  getCoordinates(T,X,Y,MAX_X,MAX_Y, p, MODE),
  replace(T,Y,X,p,B),
  verify_horizontal(B,p,X,X,Y,MAX_X,COUNT, GAME_END),
  verify_diagonals_right(B,p,Y,X,Y, MAX_Y,COUNT,GAME_END),
  verify_diagonals_left(B,p,X,Y,X,Y,MAX_X,MAX_Y,COUNT,GAME_END),
  COUNT < 5,
  display_board(B,14,2,1,0),
  playGame(B, MAX_X, MAX_Y, b, COUNT, MODE, GAME_END, 1).


playGame(T, MAX_X, MAX_Y, b, COUNT, MODE, GAME_END, VERIFY):-
  COUNT < 5,
  write('OUT OF RANGE'),nl.
```

Listing 4: **getCoordinates**

```
getCoordinates(T,X,Y,MAX_X,MAX_Y,Z,1):-
        listOfMoves(T, 0, 0, MAX_X, MAX_Y, M),
        displayList(M),nl,
        write('Y coordinate to place disk'),nl,
        read(Y2),Y1 is Y2-1,Y1<(MAX_Y),
        write('X coordinate to place disk'),nl,
        read(X2),X1 is X2-1,X1<(MAX_X),
        return_value(T,X1,Y1,R),
        R == null,
        Y is Y1,
        X is X1.


getCoordinates(T,X,Y,MAX_X,MAX_Y, b, 2):-
        random(0, MAX_Y, Y2),Y1 is Y2,Y1<(MAX_Y),
        random(0, MAX_X, X2),X1 is X2,X1<(MAX_X-Y1),
        return_value(T,X1,Y1,R),
        R == null,
        Y is Y1,
        X is X1.

getCoordinates(T,X,Y,MAX_X,MAX_Y,p, 2):-
        listOfMoves(T, 0, 0, MAX_X, MAX_Y, M),
        displayList(M),nl,
        write('Y coordinate to place disk'),nl,
        read(Y2),Y1 is Y2-1,Y1<(MAX_Y),
        write('X coordinate to place disk'),nl,
        read(X2),X1 is X2-1,X1<(MAX_X),
        return_value(T,X1,Y1,R),
        R == null,
        Y is Y1,
        X is X1.

getCoordinates(T,X,Y,MAX_X,MAX_Y,Z, 3):-
        random(0, MAX_Y, Y2),Y1 is Y2,Y1<(MAX_Y),
        random(0, MAX_X, X2),X1 is X2,X1<(MAX_X-Y1),
        return_value(T,X1,Y1,R),
        R == null,
        Y is Y1,
        X is X1.
```

Listing 5: **replace - função que coloca peça**

```
replace([L|Ls],0,X,Z,[R|Ls]):-replace_line(L,X,Z,R).
replace([L|Ls],Y,X,Z,[L|Rs]):-Y > 0, Y1 is Y-1, replace(Ls,Y1,X,Z,Rs).

replace_line([p|Cs],0,Z,[p|Cs]):-
  write('FALSE MOVE'),nl.
replace_line([b|Cs],0,Z,[b|Cs]):-
```

```
  write('FALSE MOVE'),nl.

replace_line([_|Cs],0,Z,[Z|Cs]).
replace_line([C|Cs],X,Z,[C|Rs]):-X > 0, X1 is X-1, replace_line(Cs,X1,Z,Rs).
```

Listing 6: **inteligent_coordinates**

```
updateMaxCountFound(X, Y,COUNT, FINAL_X, FINAL_Y, MAX_COUNT_FOUND, M, FX, FY):-
        COUNT > MAX_COUNT_FOUND,
        write('MAX COUNT FOUND: '),
        write(MAX_COUNT_FOUND),
        write(COUNT),nl,
        updateMaxCountFound(X, Y,COUNT, X, Y,COUNT, M, FX, FY).

updateMaxCountFound(X, Y,COUNT, FINAL_X, FINAL_Y, MAX_COUNT_FOUND, M, FX, FY):-
        M = MAX_COUNT_FOUND,
        FX = FINAL_X,
        FY = FINAL_Y,
        write('MAX COUNT:  '),
        write(M), nl.

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        COUNT < COUNT_MAX,
        X < MAX_X,
        Y < MAX_Y,
        write('AI1'),nl,
        return_value(T,X,Y,R),
        R=PLAYER,
        COUNT1 is COUNT+1,
        updateMaxCountFound(X, Y,COUNT1, FINAL_X, FINAL_Y, MAX_COUNT_FOUND,
    M, FX, FY),
        write('MAX_COUNT_FOUND  '),
        write(M),nl,
        X1 is X+1,
        write('AI1'),nl,
        intelligent_coordinates(T, PLAYER, X1, Y,RX, RY,INITIAL_MAX_X,
    MAX_X, MAX_Y, COUNT1, COUNT_MAX, FX, FY, M).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        COUNT < COUNT_MAX,
        X < MAX_X,
        Y < MAX_Y,
        X1 is X+1,
        write('AI2'),
        intelligent_coordinates(T, PLAYER, X1, Y,RX, RY,INITIAL_MAX_X,
    MAX_X, MAX_Y, 0, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
```

```
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        COUNT < COUNT_MAX,
        Y < MAX_Y,
        MAX_X1 is MAX_X-1,
        Y1 is Y+1,
        write('AI3'),
        intelligent_coordinates(T, PLAYER, 0, Y1,RX, RY,INITIAL_MAX_X,
    MAX_X1, MAX_Y, 0, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        MAX_COUNT_FOUND = 0,
        random(0, MAX_Y, Y2),Y1 is Y2,Y1<(MAX_Y),
        random(0, INITIAL_MAX_X, X2),X1 is X2,X1<(INITIAL_MAX_X-Y1),
        RY is Y1,
        write('AI33'),nl,
        RX is X1.

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X,
        MAX_X, MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        MAX_COUNT_FOUND = COUNT_MAX,
        return_value(T,FINAL_X+1,FINAL_Y,R),
        R \= p,
        R \= b,
        RX is FINAL_X+1,
        write('AI6'),
        RY is FINAL_Y.

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        MAX_COUNT_FOUND = COUNT_MAX,
        X1 is FINAL_X-COUNT_MAX,
        return_value(T,X1,FINAL_Y,R),
        R \= p,
        R \= b,
        RX is X1,
        write('AI7'),
        RY is FINAL_Y.

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X, MAX_Y,
        COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        MAX_COUNT_FOUND = COUNT_MAX,
        X1 is X+1,
        X1 < MAX_X,
        intelligent_coordinates(T, PLAYER, X1, Y,RX, RY,INITIAL_MAX_X, MAX_X,
    MAX_Y, 0, COUNT_MAX, FINAL_X, FINAL_Y, 0).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X, MAX_Y,
    COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        MAX_COUNT_FOUND = COUNT_MAX,
        Y1 is Y+1,
        Y1 < MAX_Y,
```

```
        intelligent_coordinates(T, PLAYER, X, Y1,RX, RY,INITIAL_MAX_X, MAX_X,
    MAX_Y, 0, COUNT_MAX, FINAL_X, FINAL_Y, 0).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X, MAX_Y,
    COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        write('AI9'),
        COUNT_MAX > 0,
        MAX_COUNT_FOUND1 is MAX_COUNT_FOUND,
        COUNT_MAX1 is COUNT_MAX-1,
        intelligent_coordinates(T, PLAYER, 0, 0,RX, RY,INITIAL_MAX_X,
    INITIAL_MAX_X, MAX_Y, 0, COUNT_MAX1, FINAL_X, FINAL_Y, 0).

intelligent_coordinates(T, PLAYER, X, Y,RX, RY,INITIAL_MAX_X, MAX_X,
        MAX_Y, COUNT, COUNT_MAX, FINAL_X, FINAL_Y, MAX_COUNT_FOUND):-
        COUNT_MAX = 0,
        random(0, MAX_Y, Y2),Y1 is Y2,Y1<(MAX_Y),
        random(0, INITIAL_MAX_X, X2),X1 is X2,X1<(INITIAL_MAX_X-Y1),
        RY is Y1,
        write('AI4'),
        RX is X1.
```