Xadrez Massacre

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Xadrez_Massacre_1:
Joel Dinis - ei12064

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

11 de Novembro de 2017

Resumo

O trabalho de tabuleiro que eu escolhi realizar para a cadeira de Programação em Lógica foi o Xadrez Massacre.
Este consistiu na programação de um jogo com regras semelhantes ao xadrez convencional. Para conseguir alcançar o objetivo pretendido, foi necessário o recurso a listas de listas para representação do tabuleiro, bem como mecanismos de verificação de jogadas válidas das várias peças presentes no xadrez(cavalo, bispo, rainha e torre).
Foi ainda necessário a implementação de filtragem do *input* do utilizador, providiciando o erro respetivo caso seja caso para tal. Por fim, foi preciso implementar mecanismos de deteção do estado de *gameover* e mecanismos de movimento para o bot nos vários modos(Fácil,Médio ou Difícil). Os resultados finais vieram bastante de acordo com as expectativas do projeto, sendo que o que foi dito até agora será visto nas secções abaixo com bastante mais detalhe.
Concluindo, será de esperar que se considere o projeto bem conseguido.

Conteúdo

1 Introdução

O objetivo deste trabalho foi a concretização em linguagem Prolog de o jogo existente denominado de Xadrez Massacre. Estando o objetivo concluído procedemos, pois, à descrição dos mecanismos usados para a elaboração do trabalho. No capítulo 2, é apresentada uma descrição profunda do jogo, regras e história do mesmo, pois a sua compreensão total será fundamental para perceber os passos tomados aquando da concretização da solução para este projeto.
No capítulo 3 apresentarei a forma como abordei este problema, incluindo como representei o estado de jogo, bem como a visualização para o jogador. Abordarei ainda tópicos como a validação de jogadas, como consegui implementar as regras do jogo. Ainda neste capítulo explicarei detalhadamente como o bot implementado funciona, a sua forma de avaliar as jogadas, bem como a deteção do estado de game over que permite que o jogo acabe. Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório. No capítulo 4, será abordada a interface, na maneira em como o jogador aborda a mesma e as respetivas interações. Por fim no capítulo 5, fazer-se-ão as conclusões finais acerca do projeto.

2 Xadrez Massacre

O Xadrez Massacre é uma variante do xadrez, criado por Andy Lewicki que adiciona um fator de aleatoriedade ao xadrez convencional, sendo que o tabuleiro inicial é gerado aleatoriamente, como se pode ver na figura 6. Este jogo, tal como o xadrez tradicional, é suposto ser jogado por 2 pessoas. Cada jogador possui 8 rainhas, 8 torres, 8 bispos e 8 cavalos.

Figura 1: Exemplo de um tabuleiro inicial

Cada uma das peças referidas acima possui as características do xadrez tradicional em que:

- As Torres deslocam-se horizontalmente ou verticalmente, num número ilimitado de casas, desde que estas estejam vazias(quer pelas peças do próprio jogado ou do jogador inimigo).
- Os Bispos deslocam-se na diagonal, também este num número ilimitado de casas desde que não estejam ocupadas com outras peças.
- As Rainhas podem-se deslocar na horizontal, vertical e diagonal num número ilimitado de casas, desde que o caminho para onde a peça se quer mudar esteja vazio até lá.
- Os Cavalos deslocam-se em forma de L, 2 casas para cima/esquerda/direita/baixo e posteriormente 1 para cada lado esquerda/direita. Ao contrário das outras peças, o cavalo pode pular outras peças.

Sendo assim, o tabuleiro inicial estará completamente cheio, com as 64 casas ocupadas. À medida que o jogo vai progredindo o número de peças irá sendo reduzido. Outras alterações que este jogo possui[1] são:

- Em cada jogada, é preciso que o jogador consiga eliminar uma das peças do adversário. Se este chegar a um ponto tal que não consegue eliminar peças do adversário, apesar de este ainda ter peças em campo, ele perde o jogo.
- O jogador que perder todas as peças perde o jogo.

[1] http://brainking.com/pt/GameRules?tp=128

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Como já foi referido acima, o tabuleiro inicial terá de ser gerado aleatoriamente, colocando as 64 peças de modo completamente aleatório. Sendo assim o tabuleiro será representado por uma lista de listas, em que cada lista corresponderá a uma linha do tabuleiro. No caso deste trabalho, o predicado que iniciará o estado inicial do tabuleiro será o seguinte:

```
board(B):-B=[
        [' Q ',' t ',' B ',' h ',' T ',' t ',' B ',' H '],
        [' q ',' b ',' H ',' B ',' H ',' T ',' Q ',' q '],
        [' Q ',' Q ',' T ',' h ',' h ',' T ',' b ',' b '],
        [' t ',' H ',' H ',' b ',' q ',' t ',' Q ',' B '],
        [' q ',' t ',' b ',' H ',' T ',' t ',' B ',' h '],
        [' q ',' B ',' H ',' B ',' h ',' T ',' Q ',' q '],
        [' Q ',' Q ',' h ',' H ',' T ',' B ',' t ',' b '],
        [' t ',' h ',' h ',' b ',' q ',' T ',' q ',' b ']
        ]
```

Neste tabuleiro as maiúsculas pertencem a um jogador e as minúsculas a outro e as peças existentes são Q(rainha) ,B(bispo), H(cavalo) e T(torre). Como o tabuleiro será aleatório, será necessário fazer a permutação de peças para o tabuleiro estar apto para jogo. Um exemplo de um tabuleiro inicial já permutado pode ser visto na imagem abaixo:

	0	1	2	3	4	5	6	7
0	Q	T	t	h	H	B	t	B
1	H	H	b	q	Q	q	B	T
2	h	Q	T	T	b	h	Q	b
3	H	b	H	Q	t	q	t	B
4	t	B	t	T	H	q	b	h
5	h	q	B	H	B	T	q	Q
6	Q	H	b	t	h	T	Q	B
7	T	q	q	b	h	t	h	b

Figura 2: Tabuleiro inicial permutado

À medida que o jogo vai avançando peças vão sendo capturadas, e consequentemente o tabuleiro irá ficando mais vazio. Sendo assim, a lista de listas irá parecer-se algo a este género num estado intermédio

```
[
        [' Q ',' t ',' ',' h ',' T ',' t ',' ',' H '],
        [' q ',' b ',' H ',' B ',' ',' T ',' ',' q '],
        [' ',' Q ',' T ',' h ',' h ',' T ',' b ',' b '],
        [' t ',' H ',' H ',' b ',' ',' t ',' Q ',' B '],
        [' q ',' t ',' ',' H ',' T ',' t ',' B ',' h '],
        [' q ',' B ',' H ',' B ',' h ',' T ',' ',' q '],
        [' ',' Q ',' h ',' H ',' T ',' B ',' t ',' b '],
        [' t ',' h ',' h ',' b ',' q ',' T ',' ',' b ']

]
```

No fim, há 2 possibilidades para terminar o jogo. Ou um dos jogadores não pode capturar uma das peças do adversário e perde o jogo, ou perde todas as peças e também perde o jogo. Demonstrando o último caso, o tabuleiro será parecido com algo como representado abaixo

```
[
        [' ',' ',' ',' ',' T ',' ',' ',' H '],
        [' ',' ',' ',' ',' ',' T ',' ',' '],
        [' ',' Q ',' T ',' ',' ',' T ',' ',' '],
        [' ',' ',' ',' ',' ',' ',' ',' ',' '],
        [' ',' ',' ',' ',' T ',' ',' B ',' '],
        [' ',' ',' H ',' B ',' ',' T ',' ',' '],
        [' ',' ',' ',' H ',' T ',' B ',' ',' '],
        [' ',' ',' ',' ',' ',' T ',' ',' ']

]
```

Contudo, ele também pode perder se não puder capturar peças do adversário. No exemplo abaixo, o jogador que possui as peças minúsculas não pode fazer uma jogada que resulte numa captura de uma das peças do adversário. Por isso, ele perde o jogo.

```
[
        [' t ',' ',' ',' ',' ',' ',' ',' t '],
        [' ',' ',' ',' ',' ',' T ',' ',' '],
        [' ',' Q ',' T ',' ',' ',' T ',' ',' '],
        [' t ',' ',' ',' ',' ',' ',' ',' ',' '],
        [' ',' ',' ',' ',' T ',' ',' B ',' '],
        [' ',' ',' H ',' B ',' ',' T ',' ',' '],
        [' ',' ',' ',' H ',' T ',' B ',' ',' '],
        [' ',' ',' ',' ',' ',' T ',' ',' ']

]
```

3.2 Visualização do Tabuleiro

Por forma a que o jogador possa visualizar o tabuleiro, e decidir o seu movimento serão necessários predicados para a impressão do tabuleiro. Para tal, foram construídos vários predicados em prolog para o efeito, nomeadamente

```
printab(Tabuleiro):-!,write('   0    1    2    3    4    5    6
    7'),nl, printother(Tabuleiro,-1).

printL([]).
printL([R]):-!,write(' | '),write(R),write(' | ').
printL([H|R]):-!,write(' | '),write(H),printL(R).
printother([],_).
printother(G,-1):-!,write('
    ------------------------------------------------'),nl,printother(G,0).
printother([H|R],Tab):-!,write(Tab),write(' '),printL(H),nl,Tab1 is Tab
    + 1,write('
    ------------------------------------------------'),nl,printother(R,Tab1).
```

Passo agora, pois, a explicar o programa. O predicado printab serve para identificar as colunas do tabuleiro, e chama a função responsável por imprimir o resto do mesmo. O printline é chamado e este escreve um identificador de linha, que neste caso é um número, e chama para cada elemento dentro de uma lista o predicado printL, que trata de imprimir um separador e o elemento que o identifica. Resumidamente, serve, pois o printline para imprimir uma linha e o printL para imprimir cada elemento nessa lista bem como o separador das peças. O resultado final deste código, foi já demonstrado no capítulo acima, e pode ser ainda visto na imagem abaixo:

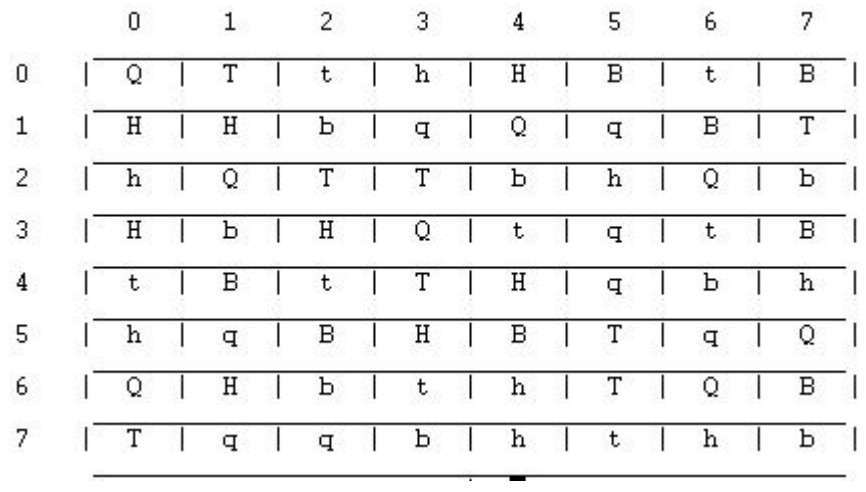

Figura 3: Tabuleiro apresentado ao jogador

3.3 Lista de Jogadas Válidas

Cada jogo possui um conjunto de regras que o torna único. O Xadrez Massacre não é diferente. Sendo este jogo uma variação do xadrez, foi preciso implementar mecanismos de movimento de peças do mesmo, nomeadamente o movimento de cavalos, bispos, rainhas e torres. Ou seja, o mecanismo de verificação de jogadas válidas deverá ser diferente se o jogador estiver a movimentar uma torre ou um bispo, por exemplo. Este mesmo mecanismo de verificação de jogadas válidas ainda verifica erros de *input*, ou erros do próprio jogador(caso este esteja a tentar comer uma das suas próprias peças, por exemplo). Passo, pois, a explicar em mais detalhe com o auxílio do código desenvolvido.

```
checkoutofbounds(Xi,Yi,Xf,Yf):-(Xf > -1, Xi > -1, Xf < 8, Xi < 8, Yf >
    -1, Yi > -1,Yf < 8, Yi < 8);(write('Out of Range'),nl,fail).

checkifsameteam(P,P2):-(((P == ' t '; P == ' h ' ; P == ' q '; P ==' b
    '),(P2 == ' T '; P2 == ' B ' ; P2 == ' Q '; P2 == ' H '));((P2 == '
    t '; P2 == ' h ' ; P2 == ' q '; P2 ==' b '),(P == ' T '; P == ' B
    '; P == ' Q '; P == ' H '')));(write('Cannot be pieces of the same
    team!!!'),nl,fail).

playerpieces(2,Xi,Yi,Tabuleiro):-!,getPeca(Xi,Yi,P,Tabuleiro),!, ((P ==
    ' t ' ; P == ' b '; P == ' h '; P == ' q ');(write('Play your own
    pieces.'),nl,fail)).
playerpieces(1,Xi,Yi,Tabuleiro):-!,getPeca(Xi,Yi,P,Tabuleiro),!, ((P ==
    ' T ' ; P == ' B '; P == ' H '; P == ' Q ');(write('Play your own
    pieces.'),nl,fail)).
```

O código acima fornecido trata de erros de *input*, quando o jogador está a tentar comer as suas próprias peças, ou se insere coordenadas fora do tabuleiro. Verifica ainda se está a tentar jogar peças do oponente. Contudo, isto não é suficiente para verificar a validade da jogada. Caso esta respeite os predicados acima mencionados, é ainda necessário verificar se a jogada é típica desse tipo de peça. Um exemplo disto é que um torre se pode movimentar horizontalmente e verticalmente, contudo um bispo só se pode movimentar diagonalmente, ou seja, dependendo do tipo de peça serão necessários mecanismos de validação diferentes.

```
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf, Xf < Xi, X2 is Xi - 1, X2
    == Xf.
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf,(Xf < Xi -> (X2 is Xi - 1,
    getPeca(X2,Yi,P,Tabuleiro), P == '
    ',clearpath(X2,Yi,Xf,Yf,Tabuleiro))).

clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1,getPeca(X2,Y2,P,Tabuleiro), P == '
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).

horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi + 1, Xf == T,T2 is Yi -
    2,Yf == T2.
```

```
validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' t '; P == ' T
    '),clearpath(Xi,Yi,Xf,Yf,Tabuleiro),!.

validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' B '; P == ' b
    '),clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro),!.

validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' H '; P == ' h
    '),horsemovement(Xi,Yi,Xf,Yf,Tabuleiro),!.

validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' Q '; P == ' q
    '),(clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro);clearpath(Xi,Yi,Xf,Yf,Tabuleiro)),!.
```

Em cima apenas estão descritas uma jogada possível de cada uma das peças, pois metendo aqui o código todo iria ocupar uma página inteira (para mais informações consultar o ficheiro). Uma torre por exemplo, apenas se pode movimentar horizontalmente ou verticalmente, e o caminho até à peça inimiga, deve estar desimpedido até lá. Ou seja, verificar apenas se a peça se move verticalmente ou horizontalmente não chega. É necessário verificar também se o caminho até lá se encontra sem peças. Neste exemplo específico, o clearpath verifica se a peça que a torre quer comer está ao seu lado(esquerdo neste caso), caso contrário verifica se apesar da peça não estar ao seu lado, o caminho está desimpedido. Contudo, e como já foi referido, aqui se encontra apenas o caso de a peça estar do lado esquerdo. O código também especifica os casos da peça estar acima, abaixo, ou direita. Em suma, este predicado verifica se o caminho para a peça que se quer comer está desimpedido e se encontra horizontalmente/verticalmente. Isto é útil nos movimentos das torres e das rainhas. O clearpathdiagonal é precisamente a mesma coisa, contudo em vez de detetar se a peça se encontra horizontalmente/verticalmente, verifica se a peça se encontra nas diagonais e se o caminho até lá também se encontra desimpedido. Isto é útil nos movimentos do bispo e da rainha. Já para o cavalo, os movimentos são bastante mais simples. Temos, pois, no predicado horsemovement, uma das 8 posições possíveis para o cavalo. Caso alguma destas condições referidas não seja satisfeita, o programa falha e volta a pedir coordenadas ao jogador.

3.4 Execução de Jogadas

Na subsecção acima tratei de explicar como se processa a validação de jogadas. Assumindo que uma jogada é valida, o programa irá tratar de a executar. Para tal, é usado o predicado abaixo.

```
replace(X,Y,P,P2,Tabuleiro,T):-append(Head,[L|Linha],Tabuleiro),
length(Head,Y),append(Antes,[P|Resto],L), length(Antes,X),
append(Antes,[P2|Resto],U),append(Head,[U|Linha],T).

makeplay(Xi,Yi,Xf,Yf,P,P2,Tabuleiro,NovoTabuleiro):-replace(Xi,Yi,P,'
    ',Tabuleiro,T),replace(Xf,Yf,P2,P,T,NovoTabuleiro).
```

O predicado makeplay, após confirmação da validade da jogada pega em ambas as peças, e ambas as coordenadas e substitui no sítio de onde a peça saiu uma peça vazia ' ', e no outro, insere a peça capturante na nova posição, devolvendo um tabuleiro novo, com as alterações realizadas, que servirá de *input* na próxima iteração do jogo. Isto tudo, validação e execução de jogada estão sempre a correr em ciclo.

```
move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro,Player):-checkoutofbounds(Xi,Yi,Xf,Yf),!,
playerpieces(Player,Xi,Yi,Tabuleiro),!,getPeca(Xi,Yi,P,Tabuleiro),
getPeca(Xf,Yf,P2,Tabuleiro),checkifsameteam(P,P2),!,
validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro),makeplay(Xi,Yi,Xf,Yf,P,P2,Tabuleiro,NovoTabuleiro).


printboard(Tabuleiro,NovoTabuleiro,Player):-repeat,printab(Tabuleiro),
getmove(Xi,Yi,Xf,Yf),move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro,Player).
```

O predicado printab imprime o tabuleiro enquanto que o getmove obtém o *input* do utilizador, e o move, trata da validação e execução da jogada.

3.5 Avaliação do Tabuleiro

A avaliação do tabuleiro, neste jogo é usada pelo bot no modo difícil. A avaliação que é feita é com base na potencialidade da próxima jogada, isto é, como este jogo é muito definido por ter sempre muitas jogadas para executar, pois a partir do momento em que não existem mais jogadas possíveis o jogador perde o jogo, este bot usa uma função para calcular a qualidade da futura jogada.

```
checklist([],NovoTabuleiro,Max,A,B,C,D,A,B,C,D,_).
checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2):-
getPeca(Xi,Yi,P,NovoTabuleiro),
getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),
findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),
(P3 == ' T ' ; P3 == ' B '; P3 == ' H '; P3 == ' Q '),
validmove(Xf,Yf,Xff,Yff,P,P3,N3),write(Xff-Yff),nl),T),
length(T,N),
(N > Max -> (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,2));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2)).

checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1):-
getPeca(Xi,Yi,P,NovoTabuleiro),
getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),
findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),
(P3 == ' t ' ; P3 == ' b '; P3 == ' h '; P3 == ' q '),
validmove(Xf,Yf,Xff,Yff,P,P3,N3),
write(Xff-Yff),nl),T),
length(T,N),
(N > Max -> (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,1));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1)).

getmovebotH(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro), (P == ' t ' ;
    P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,2).

getmovebotH(A,B,C,D,NovoTabuleiro,1):-
```

```
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro), (P == ' T ' ;
    P == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,1).
```

O bot tenta encontrar um movimento que no próximo movimento maximiza o número de futuras jogadas. Concretizando num exemplo, se o bot estiver a pensar mudar uma torre para as coordenadas X=4,y=2 ele examina quantos movimentos seriam possíveis e, com esse resultado irá comparar o número de movimentos com outras jogadas, e vai escolher aquela que, depois de escolhida, possui o maior número de alternativas/jogadas. Neste sentido, o value neste caso seria o número de futuras jogadas.

3.6 Final do Jogo

Como já foi referido, o programa funciona com um ciclo constante de *input* do utilizador. Caso o mesmo seja inválido, ele volta a pedir ao jogador novas coordenadas, se não, ele aceita a jogada, executa a mesma e consequentemente provoca uma nova iteração sobre o novo tabuleiro gerado. Contudo, a cada início de iteração é necessário comprovar se algum dos jogadores perdeu o jogo. Para tal, eu implementei o predicado gameover que indica se um determinado jogador perdeu a partida.

```
gameover(1,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),
(P == ' T ' ; P == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),
(P2 == ' t ' ; P2 == ' b '; P2 == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),
(E > 0);(printab(NovoTabuleiro),nl,write('Game Over. Player 2
    wins'),fail),!.

gameover(2,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' t ' ; P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Player 1
    wins'),fail),!.
```

O primeiro caso verifica se o jogador 1 perde o jogo. Consequentemente, o segundo caso irá verificar se o jogador 2 perde o mesmo. Este mecanismo, claro, está presente dentro do ciclo a cada nova iteração do tabuleiro.

```
cicle(Tabuleiro,2):-gameover(2,Tabuleiro),!,gameover(1,Tabuleiro),printurn(2),
printboard(Tabuleiro,NovoTabuleiro,2),changeplayer(2,P2),!,cicle(NovoTabuleiro,P2).

cicle(Tabuleiro,1):-gameover(1,Tabuleiro),!,gameover(2,Tabuleiro),printurn(1),
printboard(Tabuleiro,NovoTabuleiro,1),changeplayer(1,P2),!,cicle(NovoTabuleiro,P2).
```

3.7 Jogada do Computador

No meu trabalho eu escolhi implementar 3 dificuldades: Fácil, Médio e Difícil. Apesar de já ter abordado o modo difícil no tópico acima, irei fazê-lo novamente. O modo fácil, como o nome indica, não é muito complicado. Basicamente, este consiste em ir buscar todas as jogadas possíveis de todas as peças e escolher uma delas aleatoriamente.

```
getmovebot(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).

getmovebot(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
```

À medida que ia jogando e jogando o modo fácil fui-me apercebendo de que as peças com maior valor neste jogo seriam as rainhas, pois o facto de se poderem deslocar horizontalmente/verticalmente/diagonalmente lhes dá uma grande vantagem. Por isso decidi implementar o modo médio. Sendo assim, de todas as jogadas possíveis, se ele puder capturar uma rainha ele prioritiza essa mesma jogada. De seguida na lista das peças mais apreciadas, vêm as torres pela sua capacidade de percorrrer linhas inteiras quer na horizontal quer na vertical. Prioritizei ainda os bispos sobre os cavalos.

```
getmovebotM(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).

getmovebotM(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
```

Acima está representado um pequeno exemplo desta jogada, que representa a prioritização da captura de peças do tipo rainha, sendo que se este sucede os outros predicados de captura (cavalos, bispos, torres) não são executados(verificar programa para lista completa). Contudo, um dos problemas que sobressai com estes dois modos é que nenhum deles parece legitimamente o mínimo complexos. Por isso, decidi implementar um terceiro modo, o difícil. Este modo, ao contrário dos outros é um pouco mais ponderante na sua decisão. Ele avalia cada futura jogada como se já fosse tomada e verifica quantas jogadas possíveis teria nessa situação, sendo o valor da jogada sendo definido como a quantidade de jogadas possíveis nesta mesma situação. A jogada que apresenta um maior número de jogadas possíveis possui prioridade sobre as outras com menor número das mesmas.

```
checklist([],NovoTabuleiro,Max,A,B,C,D,A,B,C,D,_).

checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2):-
getPeca(Xi,Yi,P,NovoTabuleiro),getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),(P3
    == ' T ' ; P3 == ' B '; P3 == ' H '; P3 == ' Q
    '),validmove(Xf,Yf,Xff,Yff,P,P3,N3),T),length(T,N),(N > Max ->
    (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,2));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2)).

checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1):-
getPeca(Xi,Yi,P,NovoTabuleiro),getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),(P3
    == ' t ' ; P3 == ' b '; P3 == ' h '; P3 == ' q
    '),validmove(Xf,Yf,Xff,Yff,P,P3,N3),T),length(T,N),(N > Max ->
    (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,1));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1)).

getmovebotH(A,B,C,D,NovoTabuleiro,2):-
findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),
(P == ' t ' ; P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q '),
validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,2).

getmovebotH(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro), (P == ' T ' ;
    P == ' B '; P == ' H '; P == ' Q '),
getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2 == ' h ';
    P2 == ' q '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,1).
```

4 Interface com o Utilizador

No início do jogo é mostrado um menu ao utilizador, que escolhe se quer jogar contra um bot(pvb), contra um outro jogador(pvp), ou se apenas quer ver o bot a jogar contra si próprio (bvb). Caso o utilizador escolha uma das opções que envolvem um bot, este será ainda questionado acerca da dificuldade do bot e a partir daí o jogo começa. É ainda apresentada a vez de cada jogador, bem como o tabuleiro.

```
*--------------------*
WELCOME TO XADREZ MASSACRE!
*--------------------*

Pick a Game Mode: pvp/pvb/bvb

|: bvb.
Pick Game Dificulty: Easy(e), Medium(m), Hard(h)|:
```

Figura 4: Exemplo de interação com menu

É ainda apresentada a vez de cada jogador, bem como o tabuleiro. O tabuleiro, contudo, foi um tópico mencionado acima, pelo que não repitirei aqui.

```
It is Player 2 turn to play.
```

Figura 5: Mostra de turno

É ainda preciso uma maneira de obter as coordenadas para onde os jogadores querem capturar as peças inimigas. Para isto, implementei também mecanismos de ler input, estando o resultado presente na figura abaixo.

```
Column of the piece to move:|: 2.
Row of the piece to move: |: 0.
Column of the piece you want to move to|: 1.
Row of the piece you want to move to|: 0.
```

Figura 6: Leitura de coordenadas de peça para capturar peça inimiga

5 Conclusões

O projeto foi concluido com sucesso, sendo que as funcionalidades principais foram completamente desenvolvidas. Este foi ainda bastante útil no desenvolvimente das capacidades programacionais no que toca à linguagem Prolog. Um dos aspetos que este trabablho poderia melhor era na exploração de algoritmos mais eficientes e inteligentes da capacidade de análise de jogadas do bot.

Bibliografia

[1] Wholesale Chess. Chess pieces and how they move — wholesale chess. https://www.wholesalechess.com/pages/new-to-chess/pieces.html, 2017.

[2] Expert-Chess-Strategies. How the chess knight moves. https://www.expert-chess-strategies.com/chess-knight.html, 2008.

[3] Filip Rachunek. Brainking - regras do jogo (xadrez massacre). https://brainking.com/pt/GameRules?tp=128, 2002.

A Anexo 1

```
%5,6,7,8,9,10,12,13 A),22,25
:-use_module(library(random)). /*used to randomize which player starts*/
:-use_module(library(lists)).

%createrandtab(0,0,0,0,0,0,0,0,Tabuleiro):-write(Tabuleiro).
%createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro):- random(1,9,U),
%((U == 1,(Q1 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (Q3 is Q1 - 1, createrandtab(Q3,T1,B1,H1,Q2,T2,B2,H2,[' Q
    '|Tabuleiro]))));
%(U == 2, (T1 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (T3 is T1 - 1, createrandtab(Q1,T3,B1,H1,Q2,T2,B2,H2,[' T
    '|Tabuleiro]))));
%(U == 3, (B1 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (B3 is B1 - 1, createrandtab(Q1,T1,B3,H1,Q2,T2,B2,H2,[' B
    '|Tabuleiro]))));
%(U == 4, (H1 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (H3 is H1 - 1, createrandtab(Q1,T1,B1,H3,Q2,T2,B2,H2,[' H
    '|Tabuleiro]))));
%(U == 5, (Q2 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (Q3 is Q2 - 1, createrandtab(Q1,T1,B1,H1,Q3,T2,B2,H2,[' Q
    '|Tabuleiro]))));
%(U == 6, (T2 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (T3 is T2 - 1, createrandtab(Q1,T1,B1,H1,Q2,T3,B2,H2,[' T
    '|Tabuleiro]))));
%(U == 7, (B2 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (B3 is B2 - 1, createrandtab(Q1,T1,B1,H1,Q2,T2,B3,H2,[' B
    '|Tabuleiro]))));
%(U == 8, (H2 == 0 -> createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H2,Tabuleiro);
    (H3 is H2 - 1, createrandtab(Q1,T1,B1,H1,Q2,T2,B2,H3,[' H
    '|Tabuleiro])))).

board(B):-B=[
       [' Q ',' t ',' B ',' h ',' T ',' t ',' B ',' H '],
       [' q ',' b ',' H ',' B ',' H ',' T ',' Q ',' q '],
       [' Q ',' Q ',' T ',' h ',' h ',' T ',' b ',' b '],
       [' t ',' H ',' H ',' b ',' q ',' t ',' Q ',' B '],
       [' q ',' t ',' b ',' H ',' T ',' t ',' B ',' h '],
       [' q ',' B ',' H ',' B ',' h ',' T ',' Q ',' q '],
       [' Q ',' Q ',' h ',' H ',' T ',' B ',' t ',' b '],
       [' t ',' h ',' h ',' b ',' q ',' T ',' q ',' b ']
       ].

%imprime Tabuleiro
printab(Tabuleiro):-!,write('   0    1    2    3    4    5    6
    7'),nl, printother(Tabuleiro,-1).
printL([]).
printL([R]):-!,write(' | '),write(R),write(' | ').
printL([H|R]):-!,write(' | '),write(H),printL(R).
printother([],_).
printother(G,-1):-!,write('
    -------------------------------------------------'),nl,printother(G,0).
printother([H|R],Tab):-!,write(Tab),write(' '),printL(H),nl,Tab1 is Tab
    + 1,write('
    -------------------------------------------------'),nl,printother(R,Tab1).
```

```
%imprime vez de jogador na consola
printurn(P2):-write('It is Player '),write(P2),write(' turn to
    play.'),nl,nl.

printurnbot(2):-write('It is the bot turn to play.'),nl,nl.
printurnbot(1):-write('It is Player 1 turn to play.'),nl,nl.

printurnbotbot(2):-write('It is the Bot 2 turn to play.'),nl,nl.
printurnbotbot(1):-write('It is the Bot 1 turn to play.'),nl,nl.

%obtem peca com determinadas coordenadas
getPeca(X,Y,P,Tabuleiro):-append(Head,[L|Linha],Tabuleiro),length(Head,Y),
append(Antes,[P|Resto],L),length(Antes,X).

%muda de jogador
changeplayer(1,P2):- P2 is 2.
changeplayer(2,P2):- P2 is 1.

checkoutofbounds(Xi,Yi,Xf,Yf):-(Xf > -1, Xi > -1, Xf < 8, Xi < 8, Yf >
    -1, Yi > -1,Yf < 8, Yi < 8);(write('Out of Range'),nl,fail).
checkifsameteam(P,P2):-(((P == ' t '; P == ' h ' ; P == ' q '; P ==' b
    '),(P2 == ' T '; P2 == ' B ' ; P2 == ' Q '; P2 == ' H '));((P2 == '
    t '; P2 == ' h ' ; P2 == ' q '; P2 ==' b '),(P == ' T '; P == ' B
    '; P == ' Q '; P == ' H ')));(write('Cannot be pieces of the same
    team!!!'),nl,fail).

%verifica se o caminho se encontra desimpedido na horizontal/vertical e
    verifica se a peca se encontra nessas posicoes
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf, Xf < Xi, X2 is Xi - 1, X2
    == Xf.
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf, Xi < Xf, X2 is Xi + 1, X2
    == Xf.
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf,(Xf < Xi -> (X2 is Xi - 1,
    getPeca(X2,Yi,P,Tabuleiro), P == '
    ',clearpath(X2,Yi,Xf,Yf,Tabuleiro))).
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Yi == Yf,(Xi < Xf -> (X2 is Xi + 1,
    getPeca(X2,Yi,P,Tabuleiro), P == '
    ',clearpath(X2,Yi,Xf,Yf,Tabuleiro))).
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Xi == Xf, Yf < Yi, Y2 is Yi - 1, Y2
    == Yf.
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Xi == Xf, Yi < Yf, Y2 is Yi + 1, Y2
    == Yf.
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Xi == Xf,(Yf < Yi -> (Y2 is Yi - 1,
    getPeca(Xf,Y2,P,Tabuleiro), P == '
    ',clearpath(Xf,Y2,Xf,Yf,Tabuleiro))).
clearpath(Xi,Yi,Xf,Yf,Tabuleiro):- Xi == Xf,(Yi < Yf -> (Y2 is Yi + 1,
    getPeca(Xf,Y2,P,Tabuleiro), P == '
    ',clearpath(Xf,Y2,Xf,Yf,Tabuleiro))).

%verifica se o caminho se encontra desimpedido na diagonal e verifica se
    a peca se encontra nessas posicoes

clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1,getPeca(X2,Y2,P,Tabuleiro), P == '
```

```
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf < Xi, X2 is Xi -
    1, Y2 is Yi + 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf < Xi, X2 is Xi -
    1, Y2 is Yi + 1,getPeca(X2,Y2,P,Tabuleiro), P == '
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi > Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi - 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi > Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi - 1,getPeca(X2,Y2,P,Tabuleiro), P == '
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi < Yf, Xf > Xi, X2 is Xi +
    1, Y2 is Yi + 1,getPeca(X2,Y2,P,Tabuleiro), P == '
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi > Yf, Xf < Xi, X2 is Xi -
    1, Y2 is Yi - 1, X2 == Xf, Y2 == Yf.
clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro):- Yi > Yf, Xf < Xi, X2 is Xi -
    1, Y2 is Yi - 1,getPeca(X2,Y2,P,Tabuleiro), P == '
    ',clearpathdiagonal(X2,Y2,Xf,Yf,Tabuleiro).

%verifica se as coordenas escolhidas sao coerentes com o movimento do
    cavalo

horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi + 1, Xf == T,T2 is Yi -
    2,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi + 1, Xf == T,T2 is Yi +
    2,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi + 2, Xf == T,T2 is Yi -
    1,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi + 2, Xf == T,T2 is Yi +
    1,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi - 1, Xf == T,T2 is Yi -
    2,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi - 1, Xf == T,T2 is Yi +
    2,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi - 2, Xf == T,T2 is Yi -
    1,Yf == T2.
horsemovement(Xi,Yi,Xf,Yf,Tabuleiro):- T is Xi - 2, Xf == T,T2 is Yi +
    1,Yf == T2.


replace(X,Y,P,P2,Tabuleiro,T):-append(Head,[L|Linha],Tabuleiro),
    length(Head,Y),
append(Antes,[P|Resto],L),
    length(Antes,X),append(Antes,[P2|Resto],U),append(Head,[U|Linha],T).

%muda as pecas do tabuleiro
makeplay(Xi,Yi,Xf,Yf,P,P2,Tabuleiro,NovoTabuleiro):-replace(Xi,Yi,P,'
    ',Tabuleiro,T),replace(Xf,Yf,P2,P,T,NovoTabuleiro).

validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' t '; P == ' T
    '),clearpath(Xi,Yi,Xf,Yf,Tabuleiro),!.
validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' B '; P == ' b
    '),clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro),!.
```

```
validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' H '; P == ' h
    '),horsemovement(Xi,Yi,Xf,Yf,Tabuleiro),!.
validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro):- (P == ' Q '; P == ' q
    '),(clearpathdiagonal(Xi,Yi,Xf,Yf,Tabuleiro);clearpath(Xi,Yi,Xf,Yf,Tabuleiro)),!.

%verifica se o jogador joga as suas pecas
playerpieces(2,Xi,Yi,Tabuleiro):-!,getPeca(Xi,Yi,P,Tabuleiro),!, ((P ==
    ' t ' ; P == ' b '; P == ' h '; P == ' q ');(write('Play your own
    pieces.'),nl,fail)).
playerpieces(1,Xi,Yi,Tabuleiro):-!,getPeca(Xi,Yi,P,Tabuleiro),!, ((P ==
    ' T ' ; P == ' B '; P == ' H '; P == ' Q ');(write('Play your own
    pieces.'),nl,fail)).

%verifica gameover para player vs player
%gameover(1,NovoTabuleiro):- \+ ((getPeca(X,Y,' T
    ',NovoTabuleiro);getPeca(X,Y,' B ',NovoTabuleiro),getPeca(X,Y,' H
    ',NovoTabuleiro),getPeca(X,Y,' Q
    ',NovoTabuleiro));move(X,Y,XF,YF,NovoTabuleiro,_,1)).
gameover(1,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' T ' ; P == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Player 2
    wins'),fail),!.
gameover(2,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' t ' ; P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Player 1
    wins'),fail),!.

%verifica gameover para bot vs bot
gameoverbotbot(1,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' T ' ; P == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Bot 2
    wins'),fail),!.
gameoverbotbot(2,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' t ' ; P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Bot 1
    wins'),fail),!.
%verifica gameover para o caso player vs bot
gameoverbot(1,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' T ' ; P == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Bot wins'),fail),!.
```

```
gameoverbot(2,NovoTabuleiro):-findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P
    == ' t ' ; P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),length(Q,E),(E >
    0);(printab(NovoTabuleiro),nl,write('Game Over. Player 1
    wins'),fail),!.

%obtem movimento dos players
getmove(Xi,Yi,Xf,Yf):-write('Column of the piece to
    move:'),read(Xi),write('Row of the piece to move:
    '),read(Yi),write('Column of the piece you want to move
    to'),read(Xf),write('Row of the piece you want to move
    to'),read(Yf),nl.
shuffle(Tabuleiro,Tabuleiro,8,8).
shuffle(Tab,Tabuleiro,TabLength,Q2):-
    append(Head,[L|Linha],Tab),length(Head,Q2),random_permutation(L,F),Q3
    is Q2 +
    1,append(Head,[F|Linha],NewTab),shuffle(NewTab,Tabuleiro,TabLength,Q3).
%realiza o shuffle inicial do tabuleiro
shuffleini(Tabuleiro, TabLength,
    Q1):-board(B),append(Head,[L|Linha],B),length(Head,Q1),random_permutation(L,F),Q2
    is Q1 +
    1,append(Head,[F|Linha],O),shuffle(O,Tabuleiro,TabLength,Q2).
move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro,Player):-checkoutofbounds(Xi,Yi,Xf,Yf),!,
playerpieces(Player,Xi,Yi,Tabuleiro),!,getPeca(Xi,Yi,P,Tabuleiro),getPeca(Xf,Yf,P2,Tabuleiro),
checkifsameteam(P,P2),!,validmove(Xi,Yi,Xf,Yf,P,P2,Tabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,Tabuleiro,NovoTabuleiro).
printboard(Tabuleiro,NovoTabuleiro,Player):-repeat,printab(Tabuleiro),getmove(Xi,Yi,Xf,Yf),
move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro,Player).
%obtem movimentos por parte do bot com respetivas dificuldades
getmovebot(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebot(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).

getmovebotM(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
```

```
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' B
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' t ' ; P
    == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' H
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).

getmovebotM(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' b
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).
getmovebotM(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro),(P == ' T ' ; P
    == ' B '; P == ' H '; P == ' Q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' h
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),random_member(A-B-C-D,Q).

checklist([],NovoTabuleiro,Max,A,B,C,D,A,B,C,D,_).
checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2):-
getPeca(Xi,Yi,P,NovoTabuleiro),getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),(P3
    == ' T ' ; P3 == ' B '; P3 == ' H '; P3 == ' Q
    '),validmove(Xf,Yf,Xff,Yff,P,P3,N3)),T),length(T,N),(N > Max ->
    (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,2));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,2)).
checklist([Xi-Yi-Xf-Yf|Resto],NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1):-
getPeca(Xi,Yi,P,NovoTabuleiro),getPeca(Xf,Yf,P2,NovoTabuleiro),
makeplay(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro,N3),findall(Xff-Yff,(getPeca(Xff,Yff,P3,N3),(P3
    == ' t ' ; P3 == ' b '; P3 == ' h '; P3 == ' q
    '),validmove(Xf,Yf,Xff,Yff,P,P3,N3)),T),length(T,N),(N > Max ->
    (checklist(Resto,NovoTabuleiro,N,Xi,Yi,Xf,Yf,A,B,C,D,1));
checklist(Resto,NovoTabuleiro,Max,Xa,Ya,Xb,Yb,A,B,C,D,1)).

getmovebotH(A,B,C,D,NovoTabuleiro,2):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro), (P == ' t ' ;
    P == ' b '; P == ' h '; P == ' q
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' T ' ; P2 == ' B '; P2
    == ' H '; P2 == ' Q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,2).
getmovebotH(A,B,C,D,NovoTabuleiro,1):-
    findall(Xi-Yi-Xf-Yf,(getPeca(Xi,Yi,P,NovoTabuleiro), (P == ' T ' ;
    P == ' B '; P == ' H '; P == ' Q
```

```
    '),getPeca(Xf,Yf,P2,NovoTabuleiro),(P2 == ' t ' ; P2 == ' b '; P2
    == ' h '; P2 == ' q
    '),validmove(Xi,Yi,Xf,Yf,P,P2,NovoTabuleiro)),Q),
checklist(Q,NovoTabuleiro,-1,Xa,Ya,Xb,Yb,A,B,C,D,1).

printboardbot(Tabuleiro,NovoTabuleiro,Player,Dificulty):-printab(Tabuleiro),(Dificulty
    == 'e' -> getmovebot(Xi,Yi,Xf,Yf,Tabuleiro,Player);(Dificulty ==
    'm' -> getmovebotM(Xi,Yi,Xf,Yf,Tabuleiro,Player));(Dificulty == 'h'
    ->
    getmovebotH(Xi,Yi,Xf,Yf,Tabuleiro,Player))),move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro,Player),!.
%ciclo usado para player vs player
cicle(Tabuleiro,2):-gameover(2,Tabuleiro),!,gameover(1,Tabuleiro),printurn(2),
printboard(Tabuleiro,NovoTabuleiro,2),changeplayer(2,P2),!,cicle(NovoTabuleiro,P2).
cicle(Tabuleiro,1):-gameover(1,Tabuleiro),!,gameover(2,Tabuleiro),printurn(1),
printboard(Tabuleiro,NovoTabuleiro,1),changeplayer(1,P2),!,cicle(NovoTabuleiro,P2).
%ciclo usado para bot vs player
ciclebot(Tabuleiro,2,Dificulty):-gameoverbot(2,Tabuleiro),!,gameoverbot(1,Tabuleiro),
printurnbot(2), printboardbot(Tabuleiro,NovoTabuleiro,2,Dificulty),
changeplayer(2,P2),!,
ciclebot(NovoTabuleiro,P2,Dificulty).
ciclebot(Tabuleiro,1,Dificulty):-gameoverbot(1,Tabuleiro),!,gameoverbot(2,
    Tabuleiro),printurnbot(1),
printboard(Tabuleiro,NovoTabuleiro,1),changeplayer(1,P2),!,ciclebot(NovoTabuleiro,P2,Dificulty).
%ciclo usada para bot vs bot
ciclebotbot(Tabuleiro,Player,Dificulty):-gameoverbotbot(1,Tabuleiro),!,gameoverbotbot(2,Tabuleiro),
printurnbotbot(Player),printboardbot(Tabuleiro,NovoTabuleiro,Player,Dificulty),
changeplayer(Player,P2),!,ciclebotbot(NovoTabuleiro,P2,Dificulty).

xadrez:- shuffleini(Tabuleiro , 8 ,
    0),random(1,3,Player),cicle(Tabuleiro,Player).
bot(Dificulty):- shuffleini(Tabuleiro , 8 ,
    0),random(1,3,Player),ciclebot(Tabuleiro,Player,Dificulty).
botbot(Dificulty):- shuffleini(Tabuleiro, 8,
    0),random(1,3,Player),ciclebotbot(Tabuleiro,Player,Dificulty).
main:-botbot('H').
%inicia jogo
menu:-

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

write('WELCOME TO XADREZ MASSACRE!'),nl,

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

write('Pick a Game Mode: pvp/pvb/bvb'),nl,nl, read(Choice), (Choice ==
    'pvp' -> xadrez;Choice == 'pvb' -> (write('Pick Game Dificulty:
    Easy(e), Medium(m), Hard(h)') ,read(Input),bot(Input));Choice ==
    'bvb' ->(write('Pick Game Dificulty: Easy(e), Medium(m), Hard(h)')
    ,read(Input),botbot(Input))).
%printab(Tabuleiro),getmove(Xi,Yi,Xf,Yf),checkoutofbounds(Xi,Yi,Xf,Yf),
%move(Xi,Yi,Xf,Yf,Tabuleiro,NovoTabuleiro),cicle(NovoTabuleiro).
```