U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório Final

"Clobber"

PLOG - Trabalho prático nº1
Clobber_2

Ana Rita Norinho Pinto | up201606003

Joana Maria Cerqueira da Silva | up201208979

Índice

Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Programação Lógica, tendo como objetivo a implementação, em Linguagem Prolog, o jogo de tabuleiro Clobber. Para isso, foi usado o Sistema de Desenvolvimento SICStus Prolog como base.

Este relatório tem a seguinte estrutura:

- **Clobber:** Descrição do jogo e as suas regras.
- **Lógica do jogo:** Descrição da implementação da lógica do jogo,de acordo com a seguinte estrutura:
    - **Representação do estado do tabuleiro:** Exemplificação de estados do jogo.
    - **Visualização do tabuleiro:** Exibição do tabuleiro a partir da consola e descrição do predicado de visualização.
    - **Lista de jogadas válidas:** Obtenção de uma lista de jogadas possíveis.
    - **Execução de Jogadas:** Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo
    - **Final do Jogo:** Verificação do fim do jogo, com identificação do vencedor.
    - **Avaliação do Tabuleiro:** Forma(s) de avaliação do estado do jogo.
    - **Jogada do Computador:** Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade.
- **Interface com o utilizador**
- **Conclusão**
- **Bibliografia**

Clobber

Clobber, uma derivação do jogo Peg Duotaire, é um jogo abstrato de estratégia criado em 2001 por Michael H. Albert, J.P. Grossman e Richard Nowakowski, um grupo de entendidos em jogos combinatórios. Desde 2005 é um dos eventos da *Computer Olympiad,* uma famosa competição de programas de computadores.

É praticado por 2 jogadores que movem, alternadamente, as suas peças para uma peça de cor oposta ortogonalmente adjacente, removendo-a do jogo. No final, quem não conseguir movimentar as suas peças, perde o jogo.

Para jogar Clobber, é necessário um tabuleiro de 10x10 ou 8x8 e 2 jogadores. Este tabuleiro apresenta quadrados alternados brancos e pretos e, inicialmente, cada divisão é ocupada por uma peça da mesma cor. Para além disso, cada jogador apenas pode mover peças de uma só cor, sendo que o que move as peças brancas começa o jogo.

Referências:

https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

https://en.wikipedia.org/wiki/Clobber

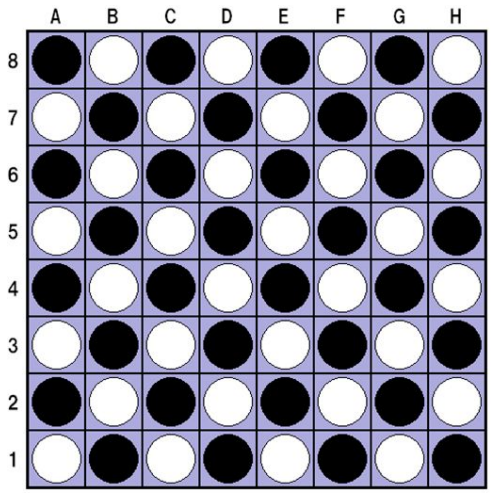


Figura 1: Estado inicial do jogo

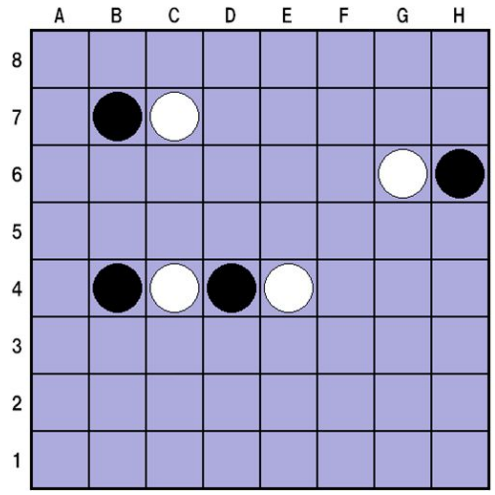


Figura 2: Estado intermédio do jogo

Como se pode observar, cada jogador tem apenas 5 possibilidades de jogada.

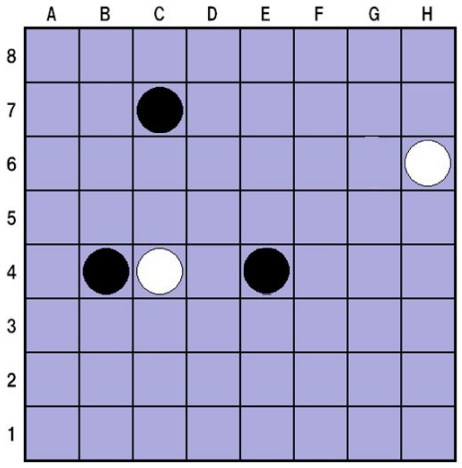


Figura 3: Penúltimo estado do jogo.
O jogador a jogar é o
que controla as peças brancas.

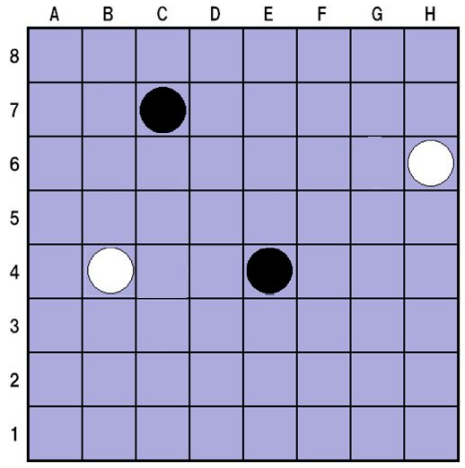


Figura 4: Estado final do jogo
É a vez do jogador que controla as peças pretas jogar.
Como não tem nenhuma jogada possível, perde o jogo.

Lógica do jogo

Representação Interna do Estado do Jogo

Estado inicial:

```
initialBoard([[black,white,black,white,black,white,black,white],
              [white,black,white,black,white,black,white,black],
              [black,white,black,white,black,white,black,white],
              [white,black,white,black,white,black,white,black],
              [black,white,black,white,black,white,black,white],
              [white,black,white,black,white,black,white,black],
              [black,white,black,white,black,white,black,white],
              [white,black,white,black,white,black,white,black],
]).
```

Estado intermédio:

```
intermediateBoard([[empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,black,white,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,white,black],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,black,white,black,white,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty]
]).
```

Estado Final:

```
finalBoard([ [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,black,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,white],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,white,empty,empty,black,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,empty]]).
```

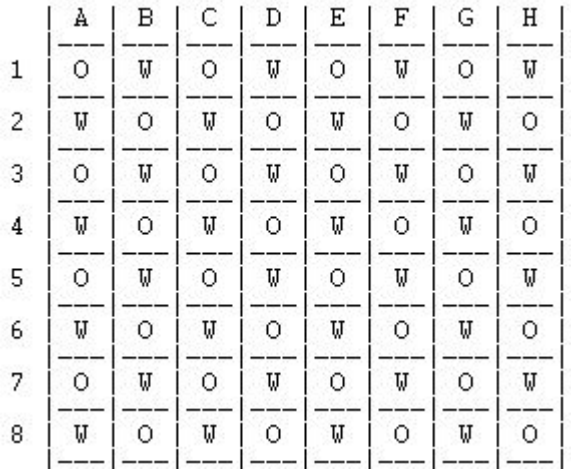


Figura 1: Estado inicial do jogo visto na consola

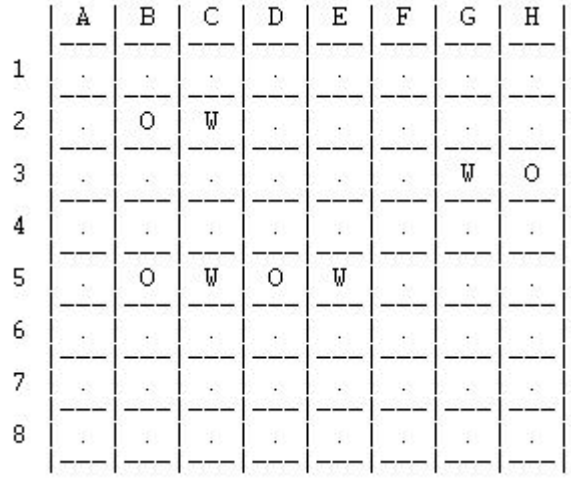


Figura 2: Estado intermédio do jogo visto na consola

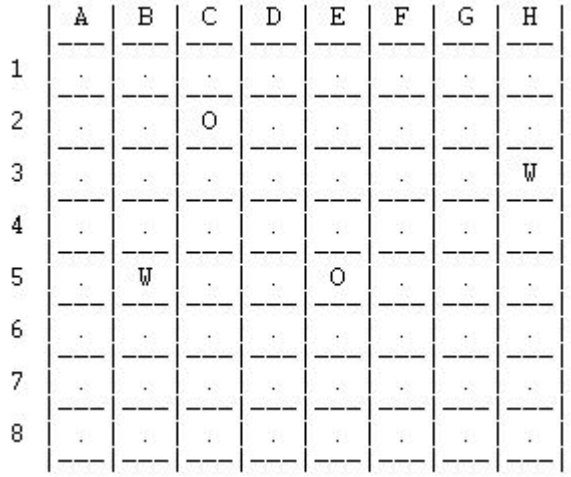


Figura 3: Estado final do jogo visto na consola

Visualização do tabuleiro

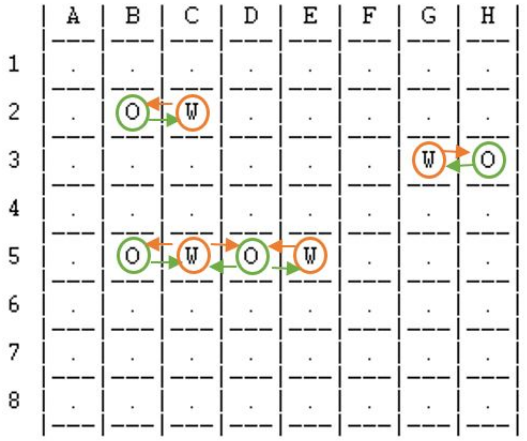
Nas imagens seguintes encontramos o código utilizado para a demonstração do tabuleiro de jogo na consola.

```
/* Regras que convertem os nomes usados no tabuleiro para um so caracter,
   de forma a tornar o jogo mais legível */
translate(empty,'.').
translate(black,'O').
translate(white,'W').

/* Incrementa o valor de X em um, usado para colocar os indices das linhas do tabuleiro */
incr(X, X1) :-
                X1 is X+1.

/* Predicado inicial que imprime o tabuleiro do jogo */
display_board([H|T]):-
                    nl,
                    write('  | A | B | C | D | E | F | G | H |\n'),
                    write('  |---|---|---|---|---|---|---|---|\n'),
                    print_tab([H|T],1).

/* Clausula responsavel por imprimir o resto do tabuleiro */
print_tab([],_).
print_tab([H|T],X):-
                  write(X),
                  write(' '),
                  write('|'),
                  write(' '),
                  print_line(H),
                  nl,
                  write('  |---|---|---|---|---|---|---|---|\n'),
                  incr(X,X1),
                  print_tab(T,X1).

/* Predicado responsavel por imprimir cada linha do tabuleiro */
print_line([]).
print_line([C|L]):-
                   print_cel(C),
                   write(' '),
                   write('|'),
                   write(' '),
                   print_line(L).

/* Predicado que imprime cada celula do tabuleiro */
print_cel(C):-
                 translate(C,V),
                  write(V).
```

Lista de jogadas válidas

Uma jogada é válida se o jogador tenta mover a sua peça para um local adjacente (cima/baixo/esquerda/direita) que tenha uma peça do adversário. Quando é pedida a introdução de uma jogada ao jogador, a posição de cada célula é dada pelo número de linha respectiva, que pode estar entre **1 e 8**, e pela letra da coluna, que se tem de encontrar num intervalo de **a-h**.

Podemos observar um exemplo no estado de jogo representado na imagem abaixo, as jogadas que cada jogador pode fazer através das setas da sua respectiva cor.



Através do predicado **valid_moves**, inserido no ficheiro *logic.pl*, é possível obter uma lista de todas as jogadas possíveis, cada uma com o formato [Row, Column, NewRow, NewColumn]. Esta cláusula chama o predicado **check_number_plays**, que percorre o tabuleiro todo e, sempre que encontra uma peça do jogador em questão, verifica em todas as direções se tem alguma possibilidade de jogada, ou seja, se se encontra uma peça do jogador adversário nos vários locais adjacentes. Esta verificação é realizada através dos seguintes predicados:

- **verify_up_move:** verifica se na célula adjacente de cima se encontra uma peça do oponente do jogador em questão e, caso seja verdade, coloca a jogada na lista;
- **verify_down_move:** verifica se na célula adjacente de baixo se encontra uma peça do oponente do jogador em questão e, caso seja verdade, coloca a jogada na lista;
- **verify_right_move:** verifica se na célula adjacente da direita se encontra uma peça do oponente do jogador em questão e, caso seja verdade, coloca a jogada na lista;
- **verify_left_move:** verifica se na célula adjacente da esquerda se encontra uma peça do oponente do jogador em questão e, caso seja verdade, coloca a jogada na lista.

Execução de jogadas

De modo a testar a validade das jogadas inseridas, foram usados os seguintes predicados, inseridos nos ficheiros *logic.pl* e *input.pl:*

- **loop_game:** Cláusula recursiva do jogo. Faz uma chamada ao predicado **ask_new_play**.
- **ask_new_play:** Verifica se é uma jogada de computador ou não, pede uma nova posição ao jogador corrente, através de **ask_new_position,** e ,certifica-se do estado de jogo atual, chamando o predicado **check_game_state**.
- **check_game_state:** Analisa o estado atual do jogo. Verifica se o jogo acabou, através do predicado **game_over**, caso falhe, passa a vez de jogada ao jogador seguinte e faz uma chamada ao **loop_game**.
- **ask_new_position:** Pede o input ao jogador da posição da peça que pretende mover e da posição para onde pretende ir, chamando depois o predicado **move**.
- **move:** Predicado que verifica se a posição atual e a nova posição são adjacentes e valida a jogada, chamando o predicado **validate_move** e o predicado **replace_in_matrix,** após o anterior se suceder.
- **validate_move:** Verifica se a posição da peça que se quer mover é uma das peças do jogador atual e, caso seja, verifica se a nova posição contém uma das peças do jogador adversário.
- **show_positions:** Mostra o movimento executado, depois de validado.
- **replace_in_matrix:** Após avaliar um movimento, substitui a posição da peça a mover (Row,Column) por "empty" e coloca-a na célula para o jogador a quer mover (NewRow, NewColumn), devolvendo em NewBoard o tabuleiro atualizado.

Final do jogo

Para avaliar se se chegou ao final do jogo, é percorrido o tabuleiro e verificado se não existe nenhuma peça do atual em células adjacentes às peças do adversário. Caso se verifique que realmente não existe mais nenhuma jogada possível, o jogador seguinte perde.

O predicado usado nesta funcionalidade é o **game_over**, contido no ficheiro *logic.pl*.

- **game_over**: enquanto o predicado value, que percorre linha a linha do tabuleiro e, sempre que encontra uma peça do jogador atual, verifica se nas células adjacentes não existe nenhuma peça do adversário, não falha, o **game_over** é executado recursivamente. Se chegar ao final do tabuleiro, significa que o jogador atual ganhou o jogo.

```
PLAYER 1
MOVEMENT
[7,h]
[7,g]
```

	A	B	C	D	E	F	G	H
1	.	.	O	.	W	.	W	.
2	O	.	.	O	.	.	.	W
3	.	O	.	.	O	.	W	.
4	W	.	.	W	.	.	.	W
5	.	.	.	.	.	W	.	.
6	O	.	.	O	O	.	.	.
7	.	W	.	.	.	.	W	.
8	W	.	.	.	O	.	O	.

```
PLAYER 2
MOVEMENT
[8,g]
[7,g]
```

	A	B	C	D	E	F	G	H
1	.	.	O	.	W	.	W	.
2	O	.	.	O	.	.	.	W
3	.	O	.	.	O	.	W	.
4	W	.	.	W	.	.	.	W
5	.	.	.	.	.	W	.	.
6	O	.	.	O	O	.	.	.
7	.	W	.	.	.	.	O	.
8	W	.	.	.	O	.	.	.

```
Player 2 won the game!!!
```

Avaliação do Tabuleiro

A avaliação do tabuleiro é realizada através dos predicados **value** e **get_value_from_matrix**, inseridos no ficheiro *logic.pl.*

- **value:** Recebe uma posição, verifica se a peça do jogador atual é igual à que se encontra nessa posição do tabuleiro, e, se for, passa à análise das peças que estão nas células adjacentes. Se não houver nenhuma peça do jogador oposto nessas, significa que não existe nenhuma jogada possível a partir dessa posição, e o predicado sucede.

  Recordando, como referido anteriormente, um jogador vence uma partida quando, após executar uma jogada, não existe nenhuma peça do adversário adjacente a uma das suas peças, visto que assim, o adversário não tem nenhuma jogada possível.

- **get_value_from_matrix:** Devolve o valor do tabuleiro na posição *Row* e *Column*.

Jogada do Computador

Foram implementadas duas formas de usar inteligência artificial no jogo, estando a funcionalidade de jogada do computador dividida em dois níveis de dificuldade: **o nível 1 e o nível 2**.

Primeiramente, no nível 1, é analisada a lista de jogadas possíveis do computador e é escolhida uma dessas jogadas aleatoriamente, através do predicado random, que gera um número aleatório entre zero e o tamanho da lista existente.

Em segundo lugar, no segundo nível, é percorrida a lista de jogadas possíveis, verificando qual delas faz com que o jogador seguinte tenha um menor número de opções de jogada. Após a escolha do movimento mais favorável para o computador, esse é executado.

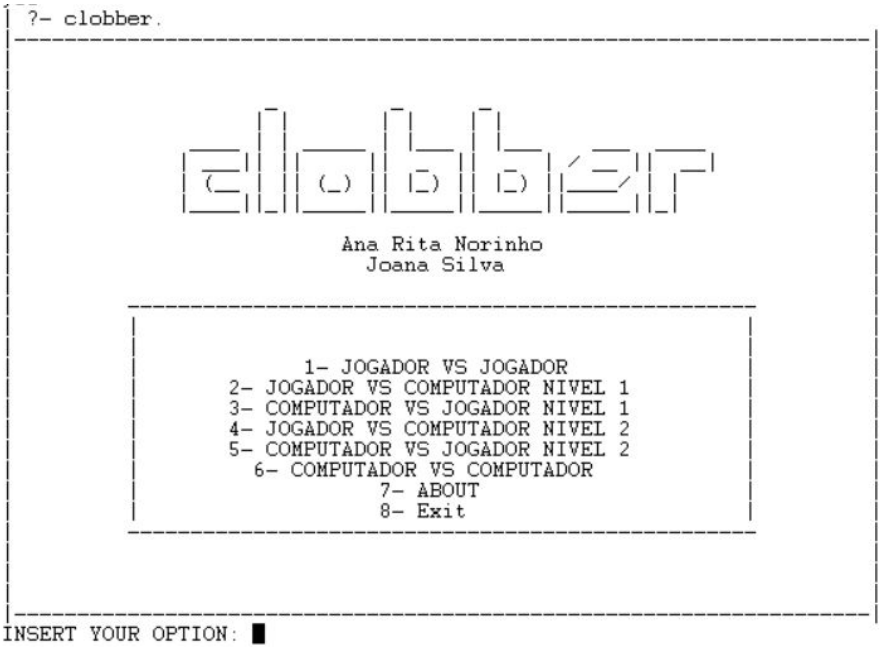
Em último lugar, é importante referir que é dada a opção de vários modos de jogo, sendo que podemos ter um jogador real a jogar contra um computador, estando os dois níveis de dificuldade disponíveis, ou apenas dois computadores a jogar. Neste último caso, na escolha de apenas termos dois computadores a jogar, o nível de dificuldade para ambos é igual ao nível um.

Os predicados usados para implementar estas funcionalidades estão descritos no ficheiro *bot.pl:*

- **generate_random_move:** verifica a lista de jogadas possíveis, gera uma posição aleatória dentro dessas jogadas.
- **generate_best_move:** responsável por gerar o movimento mais favorável para o jogador atual.

Interface com o utilizador

Ao inicializar a aplicação, através da expressão “clobber.”, na consola aparece o menu principal do jogo, que dá ao utilizador um conjunto vasto de possibilidades de escolha.



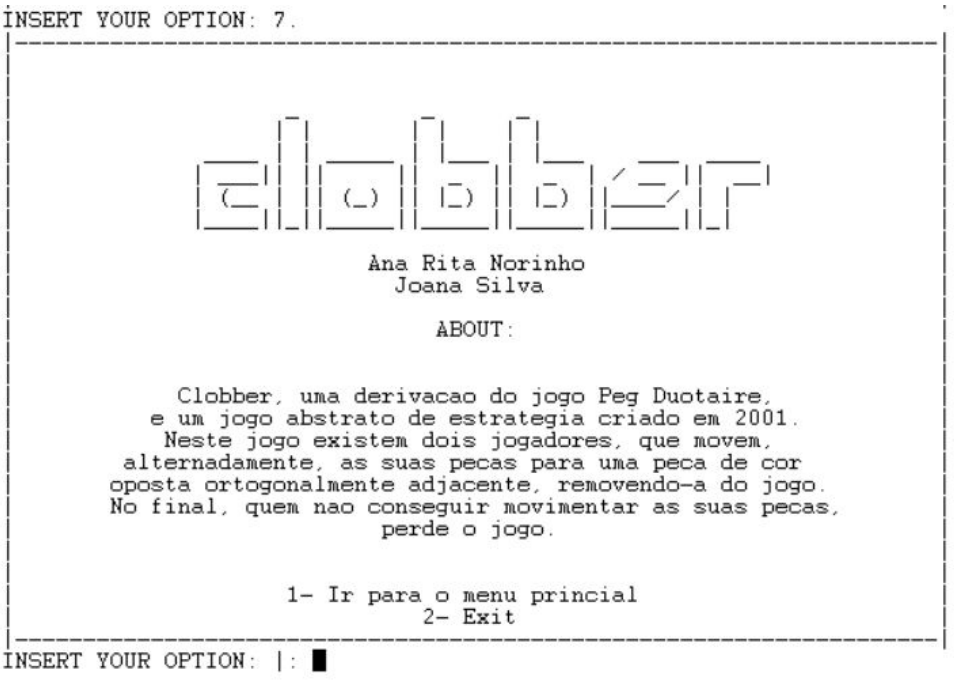
Neste menu o jogador pode escolher até oito opções, devendo este, para selecionar uma delas, colocar o número da mesma, seguido de um ponto final. Caso o número inserido não seja um dos números válidos, é pedido um novo input ao utilizador .

Na **primeira opção**, é iniciado um jogo que suporta dois jogadores. Na **segunda e terceira opções**, temos a possibilidade executar um jogo em que um jogador joga contra um computador no nível 1, ou seja, num nível em que as jogadas do computador são geradas automaticamente. A única diferença existente em ambas opções, é que na segunda, o primeiro jogador a jogar é o jogador real, apresentando este as peças brancas, enquanto que na terceira opção o computador joga primeiro.

Para as **opções quatro e cinco**, temos na mesma a possibilidade de ter uma pessoa real a jogar contra um computador, no entanto, o nível já é mais difícil, já que, em vez das jogadas do computador serem geradas automaticamente, todas as suas jogadas são realizadas segundo um cálculo que verifica qual é a melhor jogada a fazer no momento. Assim como nas opções anteriores, a única diferença entre elas ambas as opções é quem começa a jogar.

Na **sexta opção** do jogo temos a escolha do jogo ser jogado apenas pelo computador.

Por fim, o utilizador tem a opção de ir para um menu que explica as regras do jogo, como vemos na imagem em baixo. Neste menu vamos ter a opção de voltar para o menu principal e, em ambos os menus, temos também a possibilidade de sair do jogo.



Conclusões

O projeto proposto foi realizado com sucesso, tendo todas as funcionalidades pedidas. A realização do mesmo foi uma ótima forma de consolidar e praticar os conhecimentos adquiridos nas aulas, assim como também acentuar o facto da linguagem Prolog ser muito eficiente e simples em termos de problemas de decisão.

A dificuldade mais proeminente ao longo do desenvolvimento do jogo, foi o facto de estarmos a trabalhar com uma linguagem nova para nós, já que no início não nos encontrávamos muito familiarizadas com ela. No entanto, após algum tempo de habituação, as dúvidas foram desaparecendo e o resultado final obtido foi o esperado.

Bibliografia

- https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/
- https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf
- https://en.wikipedia.org/wiki/Clobber