Mestrado Integrado em Engenharia Informática e Computação

Monkey Queen

Relatório Intercalar

Programação em Lógica

Grupo Monkey_Queen_2:

Ana Rita da Costa Torres, up201406093@fe.up.pt

Rui Pedro Correia Soares, up201404965@fe.up.pt

13 de novembro de 2016

Resumo

Durante as aulas de Programação em Lógica, temos vindo a desenvolver um programa na linguagem PROLOG, nomeadamente, o jogo “Monkey Queen”.

A realização do projeto colocou o desafio de trabalhar com uma linguagem funcional, que apresenta diferenças significativas relativamente a outras linguagens mais comumente utilizadas. A solução implementada para a execução deste jogo passou pela recorrência a predicados já existentes e outros criados pelo grupo, assim como *backtracking* e algoritmos gananciosos.

Em suma, os conhecimentos relativos à linguagem em estudo aumentaram consideravelmente e o produto final é amostra disso. Acreditamos ainda, que a execução do programa na linha de comandos é simples e de interpretação fácil para o utilizador comum, permitindo oferecer uma experiência agradável.

Índice

1.Introdução

No âmbito da unidade curricular de Programação em Lógica, do curso Mestrado Integrado em Engenharia Informática e Computação foi proposto desenvolver um jogo, tendo por base a linguagem PROLOG. Das diversas opções disponíveis o grupo escolheu o "Monkey Queen", jogo da família das Damas.

O facto de o jogo ter um mecanismo percetível, embora complexo atraiu o grupo, assim como o seu tabuleiro de constituição simples, isto é, quadrado.

O objetivo principal deste relatório é explicitar toda e qualquer dúvida relativa ao código implementado para a elaboração do jogo, descrevendo a lógica deste em PROLOG, sendo a nível da representação do estado do jogo, visualização e avaliação do tabuleiro, execução de jogadas, entre outros.

2. Jogo

2.1. Descrição do Jogo

O jogo "Monkey Queen" foi concebido por Mark Steere em janeiro de 2011.

Este encontra-se disponível online nos seguintes sites:

- Ig Game Center (http://www.iggamecenter.com/info/pt/main.html)
- Mind Sports (http://www.mindsports.nl)

2.1.1 Regras do Jogo

O tabuleiro do jogo tem dimensões 12X12, cada jogador tem vinte peças em pilha, numa fase inicial, sendo um jogador da equipa *cigar* (peças castanhas) e outro da equipa *ivory* (peças brancas).

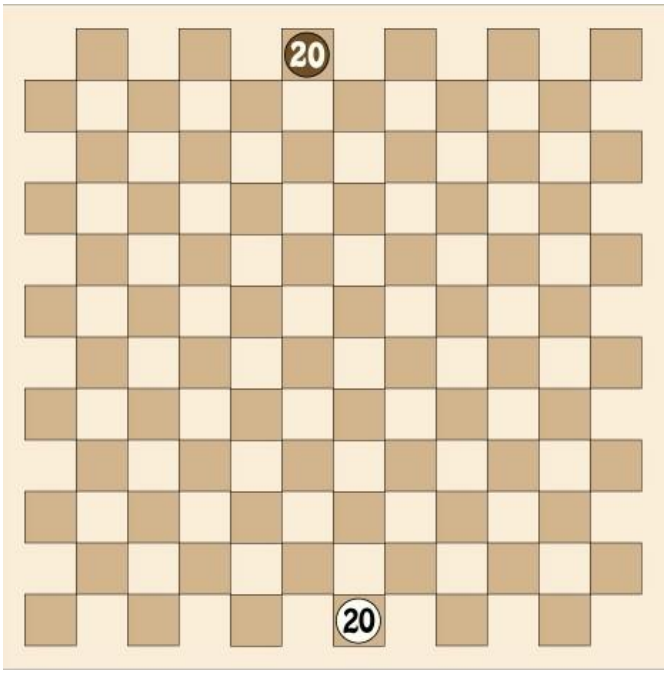

Figura 1 - Tabuleiro no Estado Inicial

O jogador da equipa *ivory* é sempre o primeiro a iniciar o jogo.

Existem dois tipos de peças:

- *Monkey Queen*: pilha de duas ou mais peças (só existe uma de cada jogador no tabuleiro);
- *Baby Monkey*: uma peça (*singleton*).

As peças movem-se em sequências de quadrados não ocupados em linha reta terminada por um quadrado ocupado pelo inimigo, podendo a orientação do movimento ser vertical, horizontal ou diagonal.

No caso da *Monkey Queen*, o movimento com captura pode ter dois desfechos, se o inimigo capturado for um *Baby Monkey*, esta ocupa o seu lugar, por outro lado se for a *Monkey Queen* adversária a equipa que realizou a jogada ganha.

Os movimentos sem captura só podem ser realizados pela *Monkey Queen*, se a sua pilha tiver um número de peças superior a dois e é obrigatório que ao efetuar o movimento esta peça deixe um *singleton* no seu quadrado de partida.

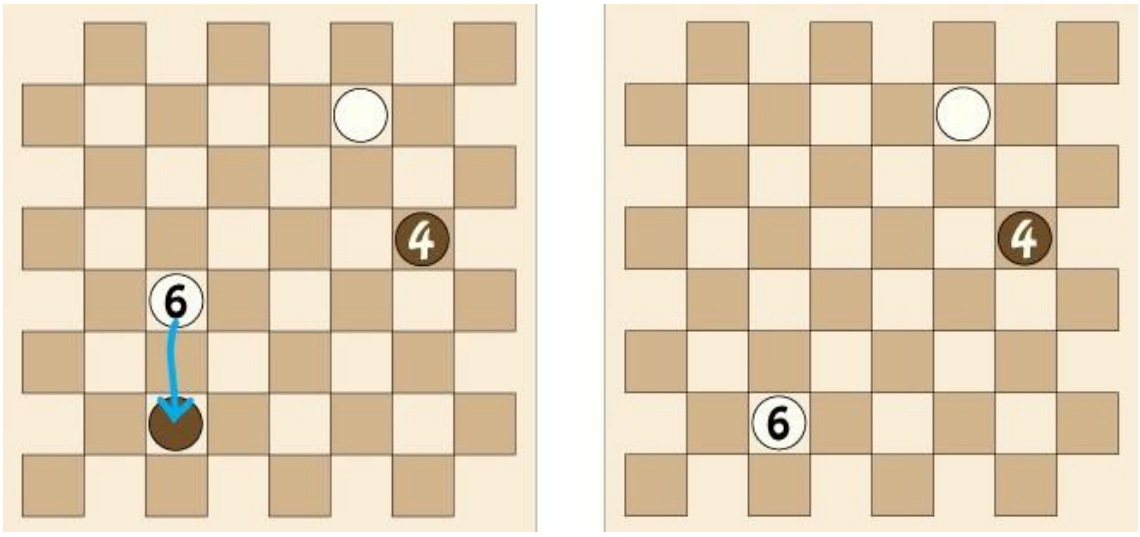

Figura 2 - Ivory Queen captura Baby Cigar

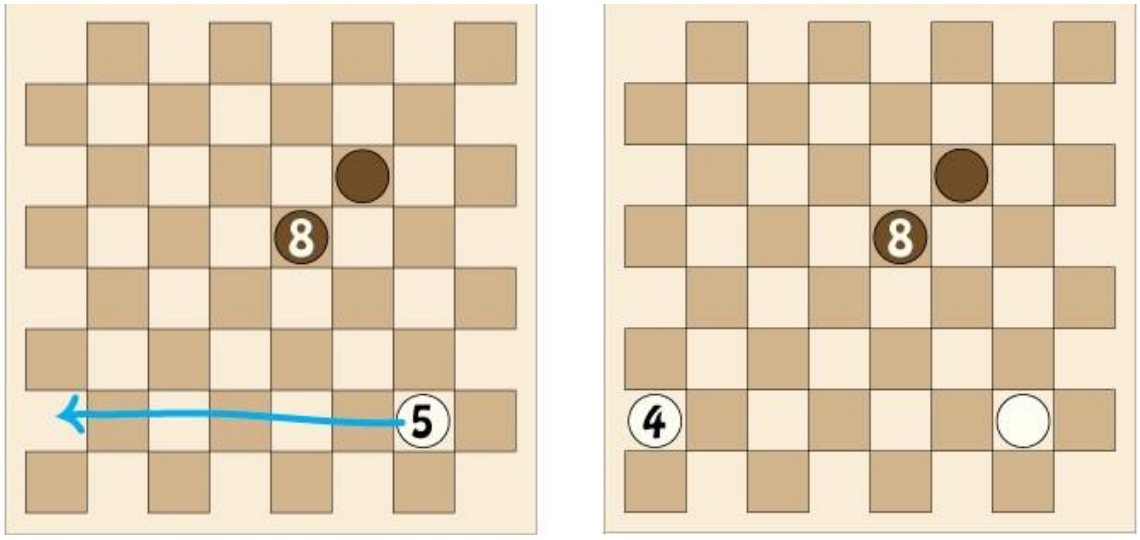

Figura 3 - Ivory Queen executa movimento sem captura

O movimento de captura das peças *Baby Monkey* é semelhante ao movimento das *Queen Monkey*. Já o seu movimento sem captura tem outras condições, ao realizá-lo é imperativo que o *Baby Monkey* se coloque numa posição que se situe a uma menor distância da *Monkey Queen* adversário do que anteriormente.

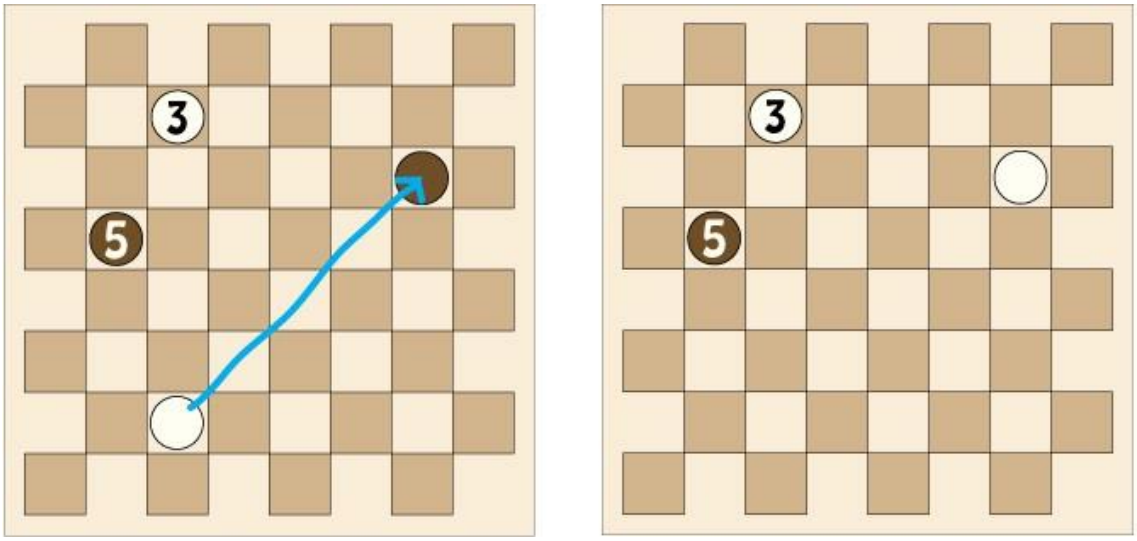

Figura 4 - Baby Ivory captura Baby Cigar

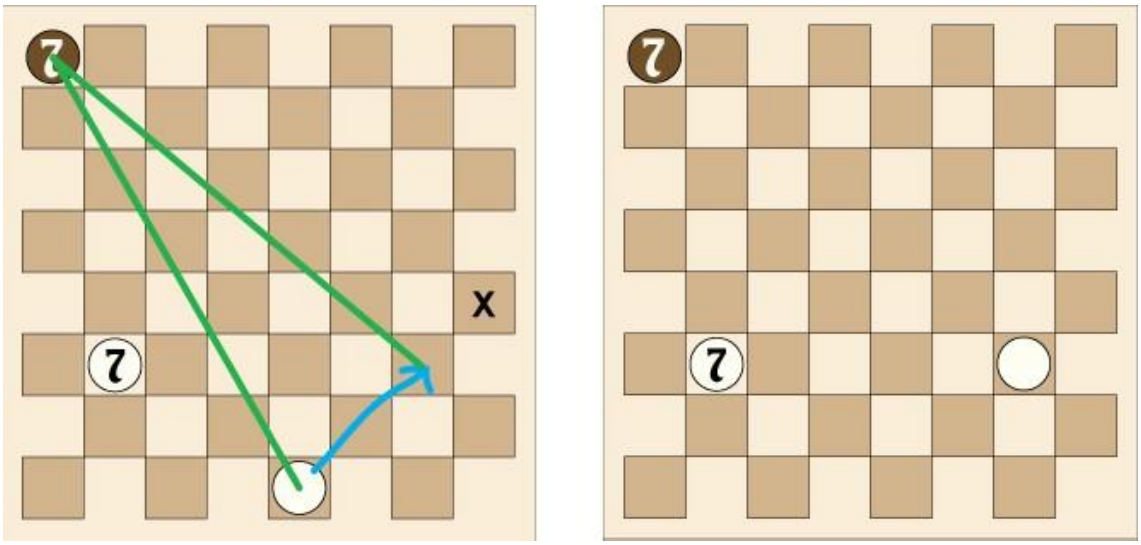

Figura 5 - Baby Ivory executa movimento sem captura

Por fim, existem 2 maneiras de ganhar o jogo, sendo que o empate é um resultado impossível neste jogo:

- Matar a *Monkey Queen* inimiga;
- Posicionar as peças de forma a que o oponente não seja capaz de realizar mais movimentos.

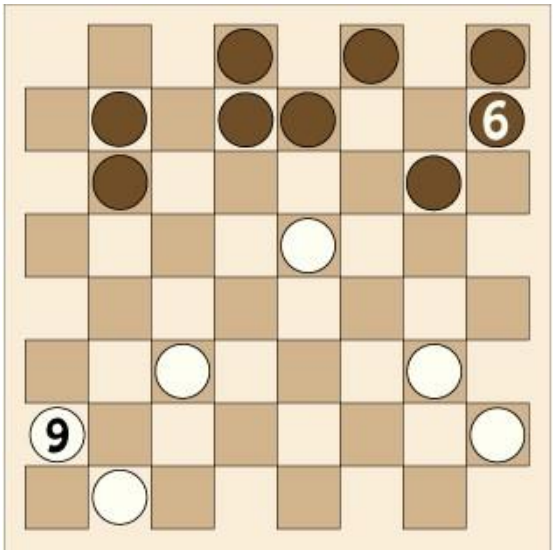

Figura 6 - Cigar Queen não consegue evitar a captura

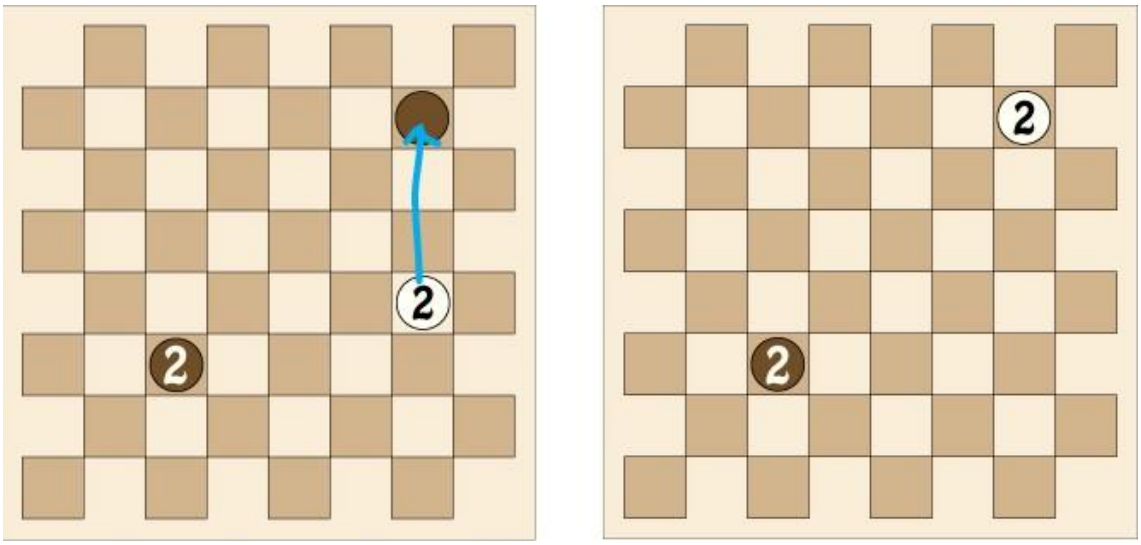

Figura 7 - Baby Ivory só tem uma hipótese de movimento, que consequentemente conduz à sua captura

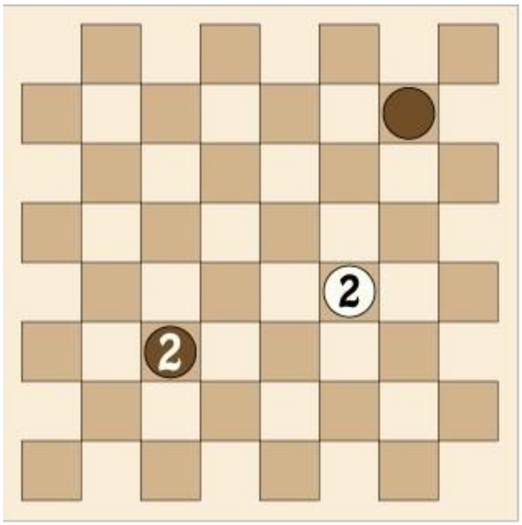

Figura 8 - Ivory Queen não pode realizar qualquer movimento e, portanto, perde o jogo

2.2. Implementação do Jogo

2.2.1. Representação do Tabuleiro

Para a implementação do jogo em PROLOG, era necessário escolher que representação de dados usaríamos para o tabuleiro e peças. Como tal, escolhemos uma matriz (ie. lista de listas), de dimensões 12x12, cujos elementos são átomos que representam o estado em que se encontra a célula. De entre os átomos possíveis, temos a célula vazia (*empty*), a Monkey Queen de cada cor (*ivoryQueen, cigarQueen*), e os seus Monkey Babies (*ivoryBaby*, *cigarBaby*).

O tabuleiro começa, tal como referido nas regras, com cada jogador com 20 peças na sua pilha inicial.

O jogador pode usar o predicado **getPiece(+X, +Y, +Board, -Symbol)** para obter a peça correspondente às coordenadas (X, Y) introduzidas. Para além disso pode usar o predicado ***setPiece(+X, +Y, +Symbol, +Board, -Board1)***.

```
emptyBoard( [[empty,empty,empty,empty,empty,ivoryQueen,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,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],
            [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,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,empty,empty,empty,empty,empty,empty],
            [empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
            [empty,empty,empty,empty,empty,empty,cigarQueen,empty,empty,empty,empty,empty]]).
```

Figura 9 - Tabuleiro Vazio

```
halfwayBoard([[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,ivoryBaby,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,empty,empty,empty,empty,empty],
              [empty,empty,empty,empty,empty,empty,empty,cigarQueen,empty,empty,empty,empty,empty],
              [empty,empty,ivoryQueen,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,cigarBaby,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,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,empty,empty,empty,empty,empty,empty,empty,empty]]).
```

Figura 10 - Tabuleiro Intermédio

```
finalBoard( [[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
             [empty,empty,empty,empty,empty,empty,cigarBaby,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,empty,empty,empty,empty,empty],
             [empty,empty,empty,empty,empty,ivoryQueen,empty,empty,empty,empty,empty,empty,empty],
             [empty,empty,cigarQueen,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,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,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,empty,empty,empty,empty,empty,empty,empty]]).
```

Figura 11 - Tabuleiro Final

2.2.2. Visualização do Tabuleiro

Para imprimir o tabuleiro de jogo para a consola, implementámos um predicado que percorre a representação do tabuleiro e imprime-o, recursivamente e fila a fila. O predicado, que para além do estado atual do tabuleiro necessita da quantidade de peças na Queen de cada jogador nesse estado é **printFancyBoard(+IvoryQueenStack, +CigarQueenStack, +Board)**.

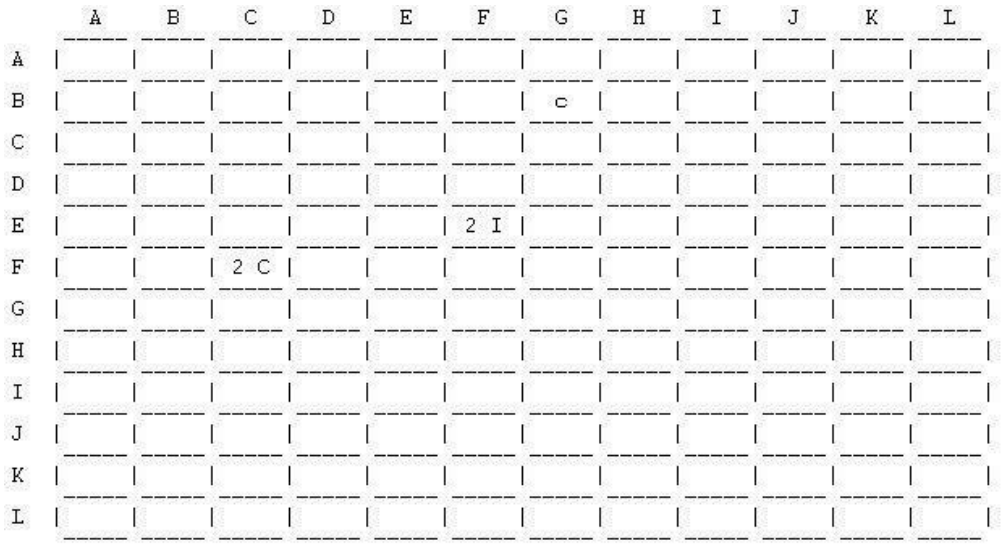

Figura 12 - Tabuleiro Inicial

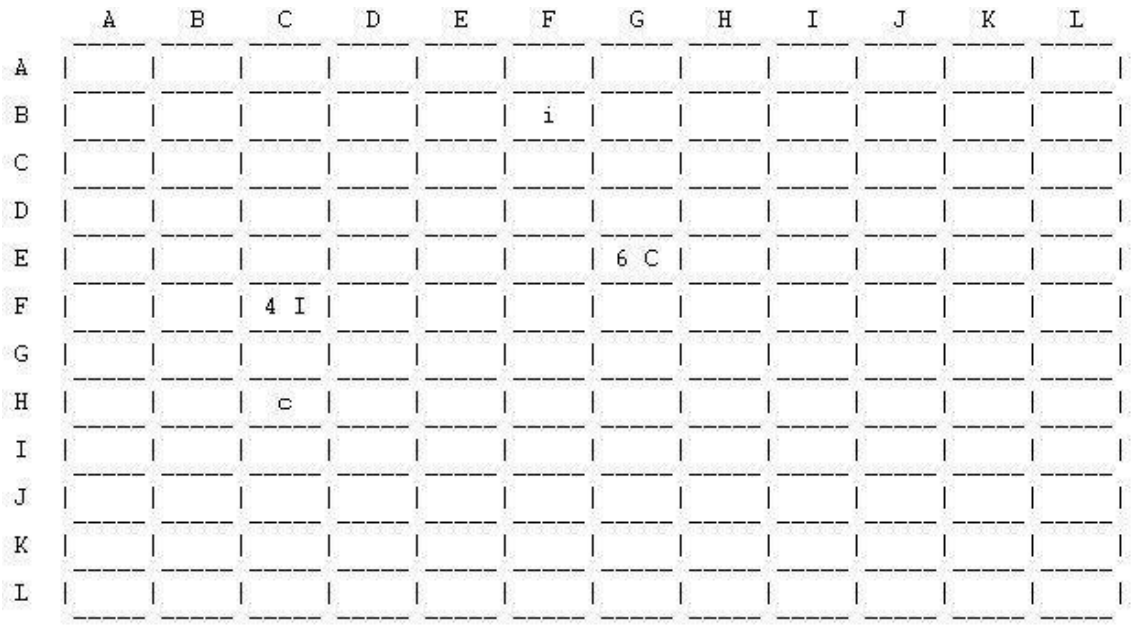

Figura 13 - Tabuleiro Intermédio

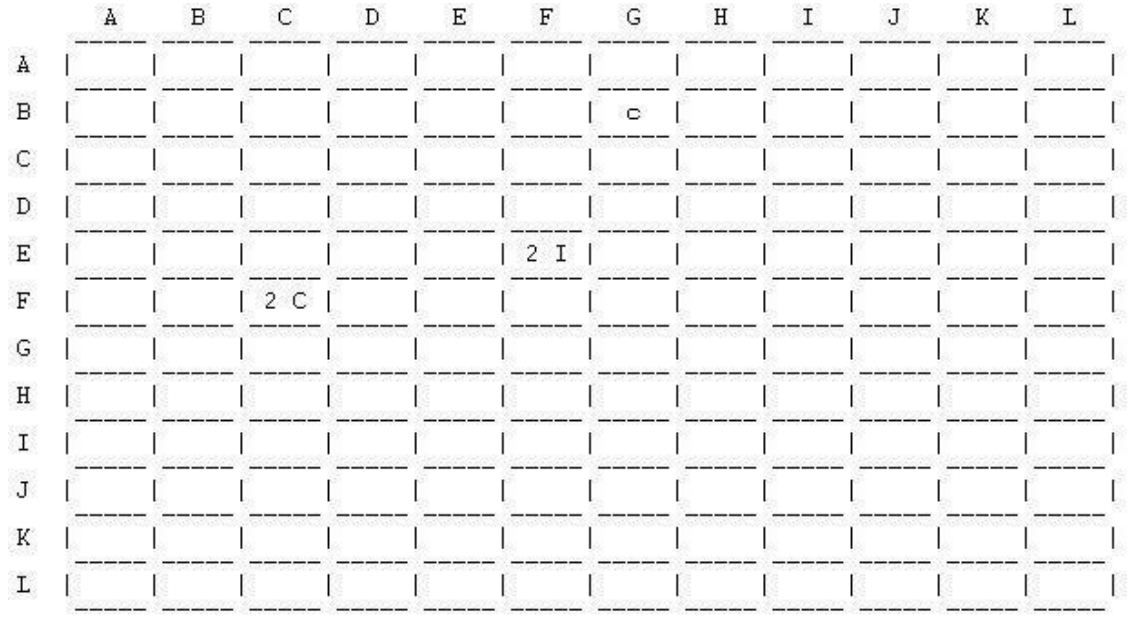

Figura 14 - Tabuleiro Final

2.2.3. Movimentos

```
makePlay((Player,X,Y,TargetX,TargetY),(IvoryStack,CigarStack,BoardIn),(IvoryStackOut, CigarStackOut, BoardOut), GameOver)
```

Figura 15 - Predicado makePlay

O predicado acima apresentado verifica se uma jogada é possível e no caso de o ser, executa o movimento. Para além disso, verifica e atualiza o estado de *Game Over*, isto é, se o jogo terminou. Este recebe o jogador, a sua posição atual, a posição para a qual este se pretende mover, a composição da Queen de cada jogador e o estado de jogo atual. Por fim, retorna a composição atualizada da Queen e o tabuleiro, após a jogada realizada. O predicado **makePlay** está no ficheiro *playMaking.pl*.

3.Lógica de Jogo

3.1. Representação do estado de jogo

```
printEmptyBoard :- Board^(emptyBoard(Board), printFancyBoard(20,20,Board)).
printHalfwayBoard :- Board^(halfwayBoard(Board), printFancyBoard(4, 6, Board)).
printFinalBoard :- Board^(finalBoard(Board), printFancyBoard(2, 2, Board)).
```

Figura 16 - Predicados de Impressão de diversos Estados de Jogo

Um estado de um jogo é o conjunto **(IvoryStack,CigarStack;Board)**,no qual *IvoryStack* e *CigarStack* são inteiros e *Board* é a matriz do tabuleiro. Todos os predicados que alteram o estado do jogo necessitam de um conjunto deste tipo.

Os predicados acima descritos representam o jogo em três estados: estado inicial, estado intermédio, isto é, um estado em jogo e o estado final, quando um dos jogadores venceu.

3.2. Visualização do tabuleiro

O tabuleiro do jogo é representado por uma lista de doze elementos, em que cada elemento é também uma lista de doze elementos, formando um tabuleiro quadrado doze por doze (12x12). As funções relativas à estrutura do tabuleiro e impressão do mesmo, encontram-se no ficheiro *printBoard.pl*. O ficheiro *boards.pl* contém apenas as representações do tabuleiro pedidas na entrega intercalar, isto é, o tabuleiro inicial, intermédio e final.

Para imprimir o tabuleiro basta chamar a função, sendo esta recursiva: **printFancyBoard(+IvoryQueenStack,+CigarQueenStack,+Board)**, em que as variáveis **IvoryQueenStack** e **CigarQueenStack** representam o número de peças de cada jogador no estado atual, sendo que, no início o valor será sempre vinte. O **Board** é um tabuleiro de um estado de jogo.

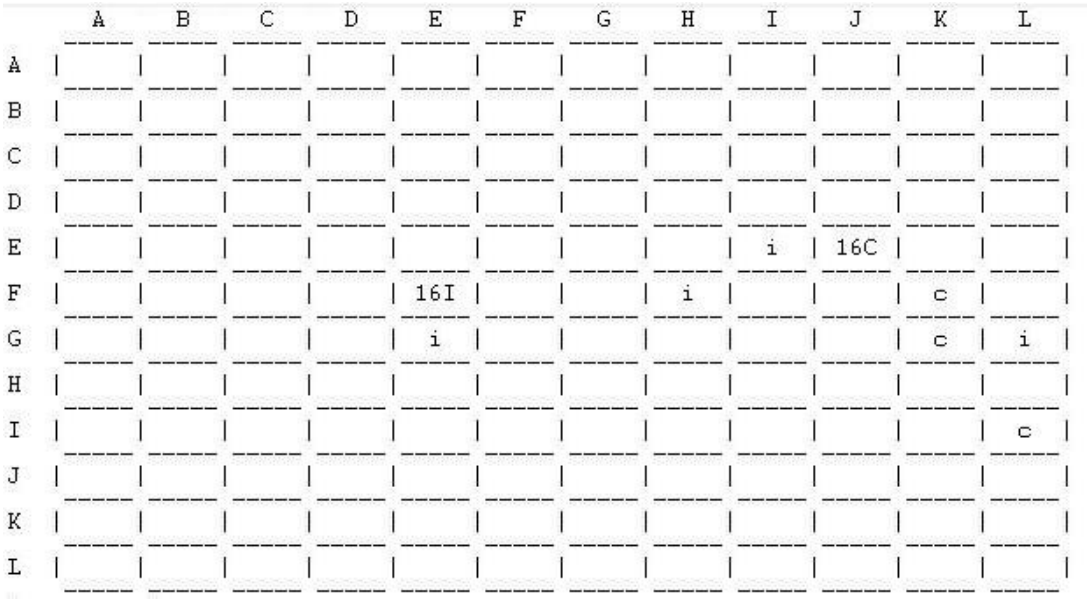

Figura 17 - Estado de jogo

3.3. Execução de jogadas

A obtenção das coordenadas de movimentação de peças é realizada por predicados distintos, dependendo do modo de jogo:

```
getPlay(Player, Play, (IvoryStack, CigarStack, Board),(IvoryStackOut, CigarStackOut, BoardOut), GameOver)
```

Figura 18 - Executa jogada de jogador humano

```
getPlayBotRandom(Player, Play, (IvoryStack, CigarStack, Board),(IvoryStackOut, CigarStackOut, BoardOut), GameOver)
```

Figura 19 - Executa jogada de Bot em modo fácil

```
getBestPlay(ivory, BoardIn, IvoryStackIn, CigarStackIn, (X,Y,TargetX,TargetY,Score))
```

Figura 20 - Executa jogada de Bot em modo difícil

No modo *Player vs Player* é realizada uma validação do jogador, seguida da validação das coordenadas atuais e alvo, depois é verificado o caminho entre a posição da peça e o seu destino, isto é, se existem outras peças pelo caminho. No caso de não existirem passa-se a uma identificação do tipo de peça e, consequentemente, à execução do movimento correto e, portanto, a jogada é aprovada.

No modo *Computer vs Computer* a maneira como as jogadas são executadas está dependente do nível de dificuldade selecionado pelo utilizador, assim como no modo *Player vs Computer*, sendo que as jogadas do *Player* são executadas como em cima explicitado. Na secção 3.7 é explicado em pormenor a execução de todas as jogadas do *bot*.

3.4. Avaliação do tabuleiro

No projeto desenvolvido, não existe uma avaliação do tabuleiro em concreto, o tipo de jogador influencia a avaliação. No caso de o jogador ser um *bot* tem-se de imediato dois tipos de avaliação, se o modo for fácil e se o modo for difícil e por fim, se o jogador for humano.

- Humano:

O jogador é livre de realizar qualquer jogada sem qualquer tipo de restrição, mesmo que não seja a mais acertada.

- *Bot* em modo fácil:

Neste modo o computador realiza os movimentos de forma aleatória e, portanto, sem qualquer tipo de condição que o restringe.

- Bot em modo difícil:

Este modo está implementado de forma a que retorne sempre a melhor jogada e, consequentemente, é o método que melhor realiza a avaliação do tabuleiro.

3.5. Final do jogo

Tal como enunciado nas regras do jogo, este termina quando ou um dos jogadores captura a Monkey Queen inimiga, ou um deles não tem mais movimentos possíveis, e acaba com a sua Queen com 2 peças. Um empate não é possível neste jogo, sendo o resultado final sempre um jogador como vencedor.

A verificação da terminação do jogo é feita no predicado

```
play(ivory, (IvoryStackIn,CigarStackIn,BoardIn), false, Type1, Type2)
```

Figura 21 - Predicado de Jogada com GameOver a false

, ou seja, no ciclo principal do jogo.

Neste, verifica se a jogada anterior terminou o jogo (se sim, *GameOver* é verdadeiro) e, se sim, imprime o ecrã de fim de jogo para o jogador que ganhou.

3.6. Jogada do computador

Neste projeto, foi-nos imposta a implementação da possibilidade do jogador defrontar um computador, e que este tivesse duas dificuldades.

Decidimos estruturar as dificuldades do seguinte modo: uma dificuldade fácil, na qual o adversário faz jogadas aleatórias, sem qualquer avaliação de resultados, e uma dificuldade mais complexa,na qual o adversário avalia qual a melhor jogada possível para o seu turno.

Para tal, foi necessário dividir as dificuldades com predicados diferentes.

- Dificuldade Easy:

Usa um método de jogada semelhante ao do jogador humano, mas em vez de precisar de *input* externo das coordenadas atuais e o alvo, gera-as aleatoriamente. Se a jogada para essas coordenadas e alvo for possível, fá-la. Senão, tenta outra vez com outras coordenadas. Esta geração de jogadas é feita no predicado

```
getPlayBotRandom(Player, Play, (IvoryStack, CigarStack, Board),(IvoryStackOut, CigarStackOut, BoardOut), GameOver)
```

Figura 22 - Geração de jogada de Bot em modo fácil

, muito semelhante ao do jogador humano. Se a jogada obtida não for possível, retorna falso.

A obtenção de jogadas até conseguir uma possível é feita no predicado que recebe

```
insistOnCorrectBotRandomPlay(Player, Play, (IvoryStackIn, CigarStackIn, BoardIn),(IvoryStackOut, CigarStackOut, BoardOut),
 GameOver):-
```

Figura 23 - Realiza repetição de jogada até esta ser aceite

os mesmos argumentos, mas repete a obtenção de jogadas até ser válida.

Nota: A obtenção aleatória de números é feita no predicado **getRandomCoords(-X)** e depende da *library random*.

- Dificuldade Hard:

Esta dificuldade é bastante mais complexa que a anterior. Usando um algoritmo ganancioso, verifica qual a melhor jogada para um estado atual do tabuleiro. O algoritmo funciona da seguinte forma:

- Percorrer todas as casas do tabuleiro. Se encontra uma peça da sua cor, guarda as coordenadas.

- Para essa peça, quer-se obter o melhor alvo. Para tal, fazer todas as jogadas possíveis, atribuir-lhes uma pontuação e retornar a melhor.

- Comparar a melhor jogada dessa peça com a melhor jogada já guardada. Se esta for melhor, passa a ser esta a jogada a retornar.

- Senão, continuar com outras peças.

- Retornar melhor jogada.

O predicado guarda a jogada na forma: **(-X,-Y,-TargetX,-TargetY,-Score)**, no qual X e Y são as coordenadas da peça escolhida, TargetX e TargetY as coordenadas-alvo, e Score a pontuação da jogada.

A atribuição de pontuação a uma jogada funciona com a seguinte ordem de prioridades:

- Come Monkey Queen inimiga (terminando o jogo);
- Come Baby Monkey inimigo;
- Mexe Baby Monkey para mais perto da Queen inimiga;
- Mexe a sua Monkey Queen(diminuindo na sua *stack*).

A obtenção da jogada ideal é feita com o predicado

```
getBestPlay(ivory, BoardIn, IvoryStackIn, CigarStackIn, (X,Y,TargetX,TargetY,Score))
```

Figura 24 - Executa jogada de Bot em modo difícil

, que efetua o algoritmo explicado em cima. Para avaliar duas jogadas, e obter a melhor das duas, é usado o

```
judgeScores((ResultingX, ResultingY, ResultingTargetX, ResultingTargetY,ResultingScore),(X1, Y1, TargetX1, TargetY1,Score1),
(_, _, _, _, Score2)) :-
```

Figura 25 - Compara jogadas e escolhe a melhor

que vê qual das jogadas tem melhor pontuação e retorna as coordenadas dessa jogada.

Depois de obter a jogada, não precisa de mais verificações, e fá-la, passando o turno ao jogador.

4.Interface com o Utilizador

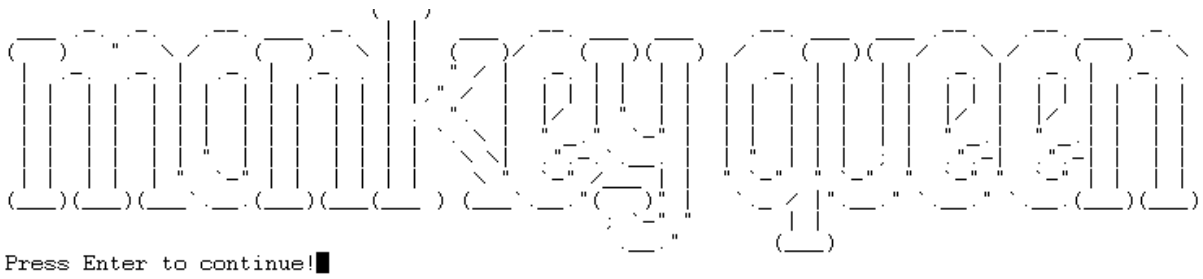

Figura 26 - Ecrã Inicial

A interface implementada permite ao utilizador jogar e usufruir do programa de forma simples. O menu inicial tem a seguinte representação:

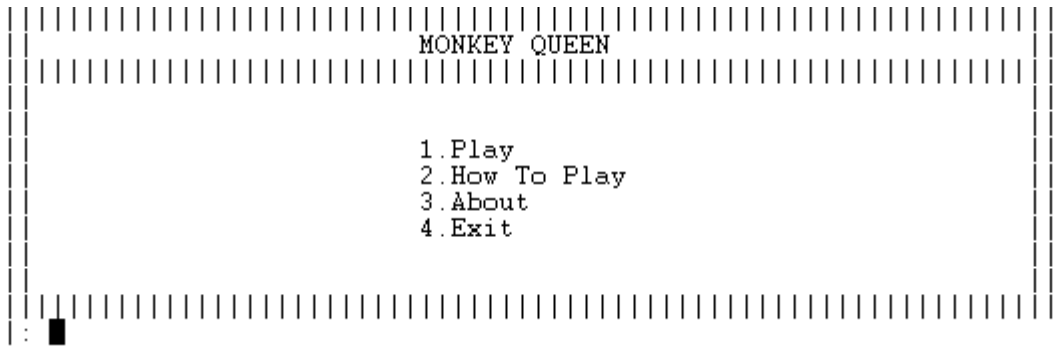

Figura 27 - Menu Principal

Ao inserir "1." na linha de comandos o utilizador é redirecionado para o menu que tem os diferentes modos de jogo:

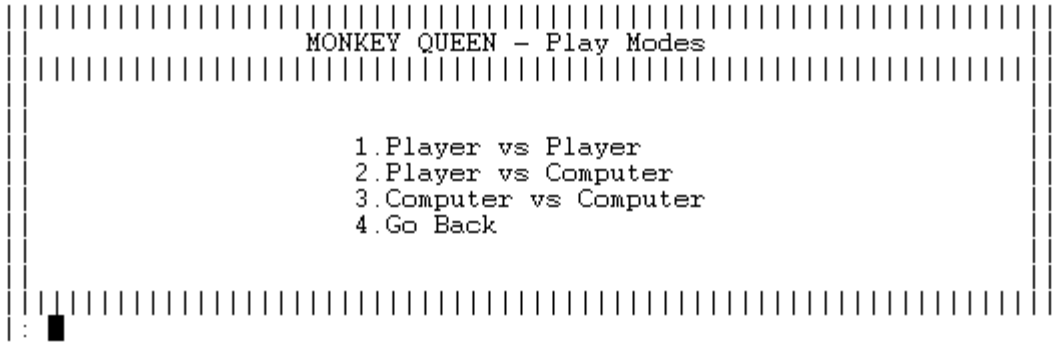

Figura 28 - Modos de Jogo

Ao inserir "1." o jogo começa de imediato, se selecionar as opções "2." ou "3." é-lhe mostrado o menu que se segue, onde o utilizador pode escolher a dificuldade do *bot*.

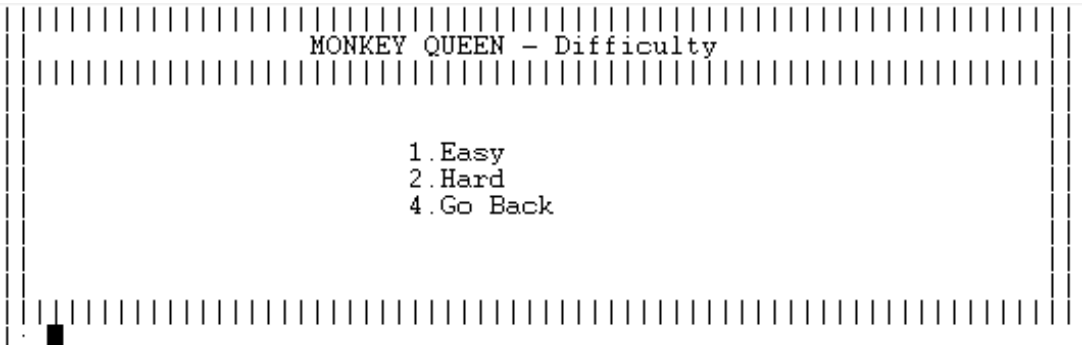

Figura 29 - Dificuldades de Jogo

Escolhendo a opção "2." do menu principal, o jogador é direcionado para um breve resumo e explicação do jogo, para passar ao painel seguinte, basta carregar na tecla *enter*.

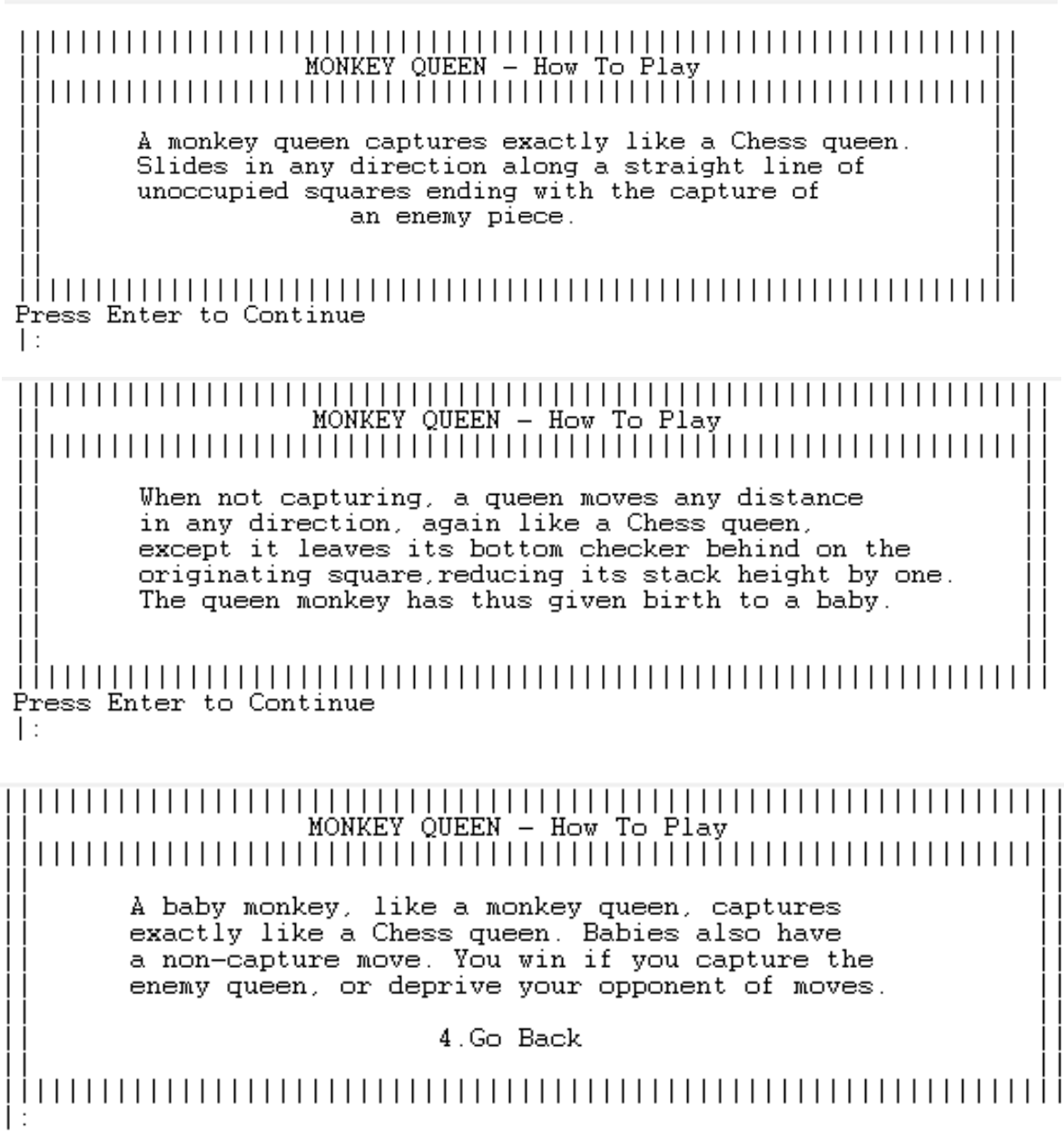

Figura 30 - Breve explicação do jogo

Se selecionar a opção "3." é redirecionado para um menu com a informação dos autores do projeto. Em qualquer um dos menus é possível retroceder, inserindo "4." na linha de comandos.

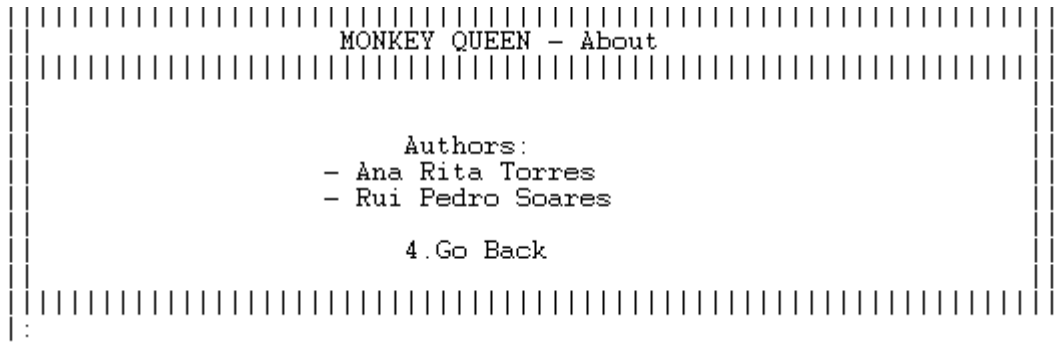

Figura 31 - Sobre

Por fim, quando o jogo termina é apresentado o estado final do jogo, com informação relativa ao vencedor.

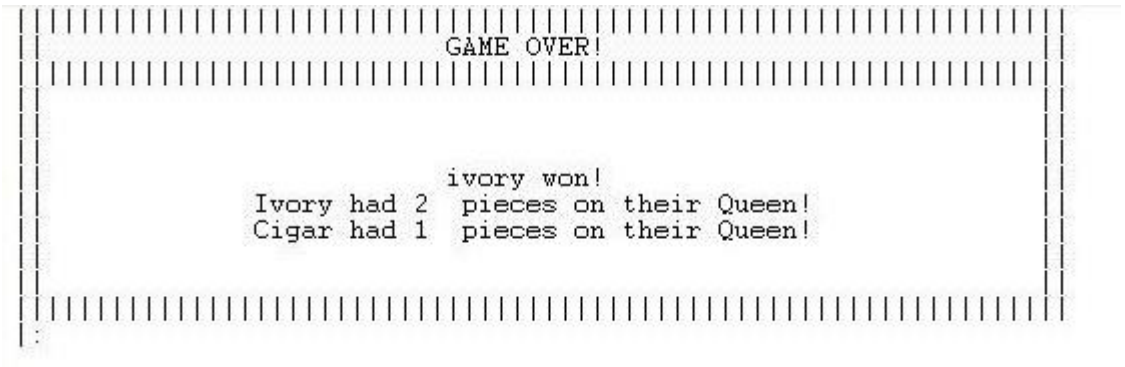

Figura 32 - Ecrã de Game Over

5.Conclusões

Apesar de exaustivo e minucioso pode-se afirmar que a realização do projeto foi fértil, na medida em que fortaleceu os nossos conhecimentos de PROLOG, tal como exercitou o nosso pensamento lógico e racional.

Considera-se que todos os objetivos de aprendizagem foram cumpridos, bem como os critérios estabelecidos, inicialmente, pelos docentes da unidade curricular.

As maiores dificuldades de desenvolvimento surgiram ao nível da implementação dos *Bots* e das respetivas dificuldades, uma vez que exigia uma maior desenvoltura e conhecimento da linguagem.

Em suma, o grupo vê o projeto desenvolvido como uma experiência enriquecedora e diferente.