Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Syrtis

GRUPO 4: SYRTIS

ANTÓNIO PEDRO ARAÚJO FRAGA (UP201303095)

CAROLINA MACEDO MOREIRA (UP201303494)

Resumo

Ao longo deste semestre temos vindo a desenvolver um jogo “Syrtis” para a cadeira de Programação em Lógica.

Sem dúvida que esta linguagem de programação não é aquilo a que estamos habituados, e este foi um dos maiores desafios no desenvolvimento deste projeto, mas que foi superado. O Syrtis é um jogo com bastantes pormenores, mas pensamos que conseguimos implementá-los a todos, com uma interface *user-friendly* e com uma verificação de erros bastante completa.

Durante o desenvolvimento deste projeto foi necessário a implementação de algoritmos um pouco mais complexos como o algoritmo de reconhecimento de ilhas. Este é um algoritmo que reconhece se duas casas (com uma dada posição [X1, Y1] e [X2, Y2]) se encontram na mesma ilha, seja ela composta por qualquer tipo de peça.

Depois de troca de impressões sobre conhecimentos lecionados em PLOG, ficamos com a sensação que há projetos bastante mais simples, contudo este desafio foi bastante bem-vindo mas com um prazo final um pouco apertado, e visto que é dificil obter todas as jogadas possíveis num jogo como este, vimo-nos obrigados a não concluir a parte “jogador vs computador”.

Índice

Introdução

Na cadeira de PLOG do curso de Mestrado Integrado em Engenharia Informática e Computação foi-nos dado o desafio de desenvolver um jogo de tabuleiro em PROLOG. A escolha deste jogo foi baseada no desafio que este projeto representa para nós, um jogo com uma história bastante interessante e um jogo muito promenorizado. Os nosso objetivos seriam completar todos os pontos propostos pelos docentes da cadeira e apresentar uma jogabilidade simples, praticando ao mesmo tempo todos os conhecimentos adquiridos nas aulas teóricas.

A estrura deste relatório encontra-se dividida em vários pontos cruciais com toda a informação sobre o desenvolvimento deste projeto:

- História e regras do jogo
- Lógica implementada
- Interface
- Conclusões

O jogo Syrtis

História

O ***Syrtis*** é um jogo de dois jogadores inspirado nas areias movediças do Mar Mediterrâneo, foi concebido e desenhado por David Vander Laan em 2014, um jogo que se enquadra na categoria de Estratégia Abstracta.

Constituição do tabuleiro

Este jogo tem um tabuleiro com 36 ou 16 "casas", casas essas que podem ser de quatro tipos, circulares ou quadradas, com possibilidade de uma coloração clara ou escura. As peças utilizadas são constituidas por quatro torres, duas redondas, claras, e duas quadradas, escuras.

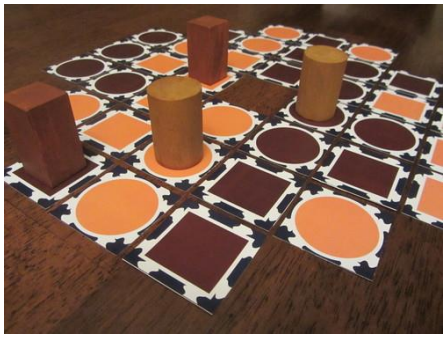

Figura 1 (um exemplo de um tabuleiro de jogo)

Distribuição de casas

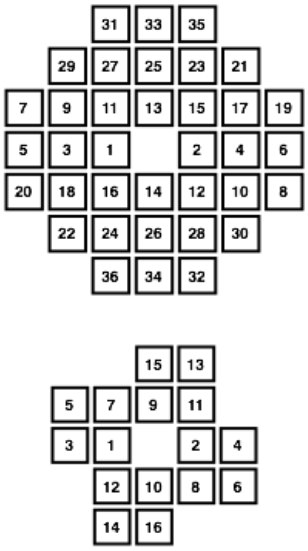

Figura 2 (disposição de casas)

Inicialmente é escolhida aleatoriamente uma casa para ser colocada na posição 1. Depois disso, é escolhida outra casa de cor e forma oposta para ser colocada na posição 2. O processo de escolha aleatória para as posições ímpares e a escolha de uma casa com cor e forma oposta para as posições pares continua até o tabuleiro estar completo, criando-se assim um tabuleiro anti-simétrico.

Um dos jogadores deve escolher a posição inicial das torres, cada uma delas deve estar situada numa casa desocupada com a mesma forma e cor. O jogador que não escolheu a disposição das peças, deve escolher com que peças jogar, claras ou escuras, as peças claras jogam primeiro.

Para jogos mais rápidos são usadas 4 casas de cada tipo, e deve-se seguir a disposição da figura da esquerda.

O que são ilhas?

Uma ilha é uma casa ou um conjunto de casas que têm a mesma forma ou a mesma coloração (casas que se apresentem na diagonal não fazem parte da ilha). Há assim quatro tipos de ilhas, ilhas de casas redondas, de casas quadradas, de casas claras e de casas escuras.

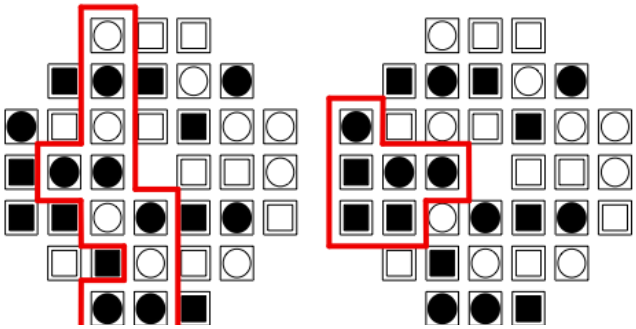

Figura 3

Movimentos

Um jogador pode fazer exatamente um movimento: **mover uma torre**, **afundar uma casa**, **mover uma casa** ou **passar**. No fim de cada jogada, todas as casas devem estar interligadas.

Mover uma torre: Se estás a jogar com peças claras, podes mover uma torre clara para qualquer casa desocupada de uma ilha de cor clara ou uma ilha de casas circulares em que a peça se encontre. Se estás a jogar com peças escuras, podes mover a torre para qualquer casa

desocupada de uma ilha de casas escuras ou uma ilha de casas quadradas em que a peça se encontre.

Afundar uma casa: (1) Um jogador pode remover uma casa que esteja adjacente a uma casa onde esteja situada uma das peças do jogador. (2) Um jogador pode afundar uma casa desocupada. (3) Um jogador pode afundar uma casa que tenha pelo menos um lado aberto, ou seja, uma casa que não tenha uma casa vizinha em qualquer um dos lados. (*Relembrar que as casas devem estar interligadas ao final de cada ronda*)

Mover uma casa: Um jogador pode mover uma casa ocupada por uma das suas torres através de qualquer número de espaços vazios interligados, em qualquer número de direções. Quando um jogador move uma casa não pode aumentar a altura ou largura do tabuleiro. Por altura e largura do tabuleiro entenda-se caixa delimitada na figura à direita.

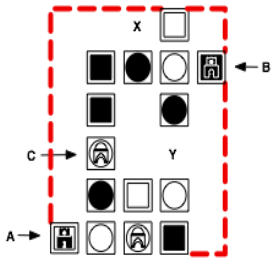

Figura 4

Passar: Um jogador pode passar quando bem entender, mas é obrigado a passar se não pode tomar mais nenhuma ação.

Fim do jogo

Há três possibilidades de chegar ao fim do jogo:

Completar uma ilha: Um jogador é vencedor se todas as casas com cor ou forma igual à das suas torres estiverem ligadas entre si. Se isto acontecer quando o tabuleiro é inicialmente construído, o jogador que coloca as torres no tabuleiro deve trocar as casas entre os pares de posições que estas são incialmente colocadas ex: (11 e 12) (7 e 8) (25 e 26).

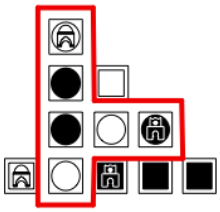

Figura 5 (uma situação de vitória para as peças claras)

Afundar peças: Um jogador é vencedor se afundar quatro casas sem que o jogador oponente afunde alguma. Perde assim uma corrida contra areias movediças.

Iniciativa: Esta regra impede empates. O jogo termina em qualquer uma das seguintes situações:

- Ambos os jogadores completam uma ilha ao mesmo tempo.
- Os jogadores passam em turnos consecutivos.
- Um jogador passa 4 vezes consecutivas.

Nesta situação, o jogador vencedor é aquele que tem mais peças afundadas desde que o outro jogador tenha afundado alguma peça. Numa situação improvável em que nenhuma peça tenha sido afundada, ganha o jogador que jogou primeiro.

Lógica de Jogo

Representação do estado de jogo

Para uma melhor jogabilidade, é necessário ter conhecimento da cor e a forma das casas em que as peças (torres) se encontram, e como tal decidimos seguir o conselho do professor e usar o SWI-Prolog, tornando-se possível dar coloração aos caracteres.

No jogo original, as peças têm uma coloração clara e escura, mas para uma melhor visualização na consola, trocamos a cor clara por azul e a cor escura por vermelho.

Representação de um estado inicial do tabuleiro

L0 = ['vazio', 'vazio', 'o-azul', 'quadrado-vermelho', 'o-azul', 'vazio', 'vazio'],

L1 = ['vazio', 'quadrado-vermelho', 'o-azul', 'quadrado-vermelho', 'o-vermelho', 'quadrado-vermelho', 'vazio'],

L2 = ['o-vermelho', 'o-azul', 'quadrado-azul', 'quadrado-vermelho', 'o-vermelho', 'quadrado-azul', 'quadrado-azul'],

L3 = ['o-vermelho', 'quadrado-azul', 'quadrado-azul', 'vazio', 'o-vermelho', 'o-vermelho', 'quadrado-azul'],

L4 = ['o-vermelho', 'o-vermelho', 'quadrado-azul', 'o-azul', 'o-vermelho', 'quadrado-vermelho', 'quadrado-azul'],

L5 = ['vazio', 'o-azul', 'quadrado-azul', 'o-azul', 'quadrado-vermelho', 'o-azul', 'vazio'],

L6 = ['vazio', 'vazio', 'quadrado-vermelho', 'o-azul', 'quadrado-vermelho', 'vazio', 'vazio'],

Tabuleiro = [L0, L1, L2, L3, L4, L5, L6].

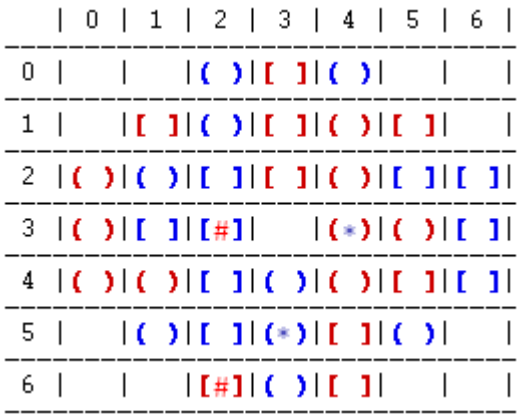

Figura 6 (representação de um possível estado inicial)

Representação de um estado intermedio do tabuleiro

L0 = ['vazio', 'vazio', 'vazio', 'vazio', 'quadrado-azul', 'vazio', 'vazio'],

L1 = ['vazio', 'vazio', 'quadrado-vermelho', 'o-vermelho', 'o-azul', 'quadrado-vermelho', 'vazio'],

L2 = ['vazio', 'vazio', 'quadrado-vermelho', 'vazio', 'o-vermelho', 'vazio', 'vazio'],

L3 = ['vazio', 'vazio', 'quadrado-azul', 'vazio', 'vazio', 'vazio', 'vazio'],

L4 = ['vazio', 'vazio', 'o-vermelho', 'quadrado-azul', 'o-azul', 'vazio', 'vazio'],

L5 = ['vazio', 'quadrado-azul', 'o-azul', 'o-azul', 'quadrado-vermelho', 'vazio', 'vazio'],

L6 = ['vazio', 'vazio', 'vazio', 'vazio', 'vazio', 'vazio', 'vazio'],

Tabuleiro = [L0, L1, L2, L3, L4, L5, L6].

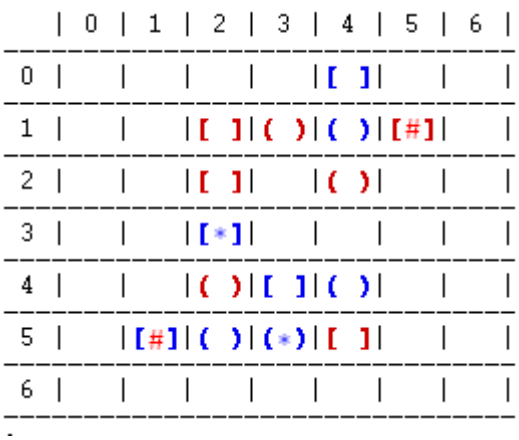

Figura 7 (Um possível estado intermédio)

Representação de um estado final do tabuleiro

L0 = ['vazio', 'vazio', 'vazio', 'vazio', 'quadrado-azul', 'vazio', 'vazio'],

L1 = ['vazio', 'vazio', 'quadrado-vermelho', 'o-vermelho', 'o-azul', 'quadrado-vermelho', 'vazio'],

L2 = ['vazio', 'vazio', 'quadrado-vermelho', 'vazio', 'o-vermelho', 'vazio', 'vazio'],

L3 = ['vazio', 'vazio', 'quadrado-azul', 'vazio', 'vazio', 'vazio', 'vazio'],

L4 = ['vazio', 'vazio', 'o-vermelho', 'quadrado-azul', 'o-azul', 'vazio', 'vazio'],

L5 = ['vazio', 'quadrado-azul', 'o-azul', 'o-azul', 'quadrado-vermelho', 'vazio', 'vazio'],

L6 = ['vazio', 'vazio', 'vazio', 'vazio', 'vazio', 'vazio', 'vazio'],

Tabuleiro = [L0, L1, L2, L3, L4, L5, L6].

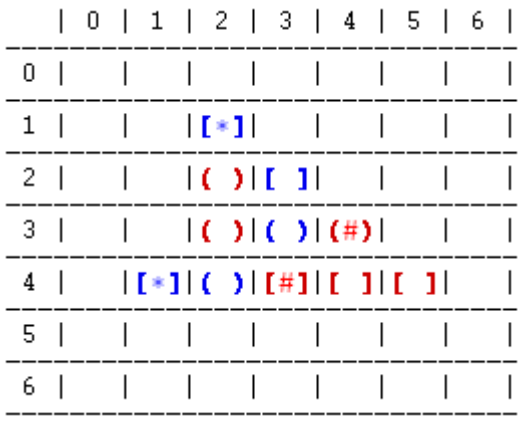

Figura 8 (Um possível estado final)

Visualização do tabuleiro

```
imprimeTabuleiro([Cabeca | Cauda], X, Tamanho):-
        X < Tamanho,
        imprimeNCol(X, 0, Tamanho),
        imprimeLinha(Cabeca, X, 0, Tamanho),
        imprimeSeparador(-1, Tamanho), %--desde o indice -1, para cobrir os numeros das linhas--%
        X1 is X + 1,
        imprimeTabuleiro(Cauda, X1, Tamanho).
```

Este é o predicado para imprimir o tabuleiro de jogo, e um dos outputs esperados seria o conteúdo apresentado na Figura 6.

Execução de jogadas e Final do Jogo

A execução de jogadas passa por três fases, no ínicio é chamado um predicado **jogo(+Tabuleiro, +Jogador),** para que o jogador possa escolher o movimento a ser executado e todos os seus promenores. Depois disso é chamada uma verificação de jogada válida com predicados adequados à jogada escolhida anteriormente:

- moveTorre(+Tabuleiro, +Jogador, +X, +Y, +XFinal, +YFinal, +XLimite, +Ylimite)
- afundaCasa(+Tabuleiro, +Jogador, +X, +Y, +XLimite, +Ylimite)
- moveCasa(+Tabuleiro, +Jogador, +X, +Y, +XFinal, +YFinal, +XLimite, +Ylimite)

A execução do predicado **passa** não necessita de uma validação, portanto não tem esta Fase.

Depois da validação de jogada, é executada a verificação de fim de jogo, que é um *overload* do predicado **jogo**, chamado inicialmente, o jogo continuará se essa verificação falhar.

Quando uma jogada é bem sucedida, é chamado um predicado de troca de jogador **trocaJogador(+Jogador, -JogadorTrocado).**

Como referido anteriormente foram desenvolvidos vários algoritmos, um pouco mais complexos que o habitual, para validação de jogadas e para criar um tabuleiro aleatório anti-simétrico.

Nota: Há mais argumentos que compõe os predicados descritos em cima, argumentos que são utilizados para verificação do estado final do jogo, como o número de vezes consecutivas que o utilizador passa a jogada e o número de vezes consecutivas que o jogador afundou uma casa.

Interface com o utilizador

Durante o desenvolvimento deste projeto, concluímos que seria melhor organizar o código desenvolvido em vários ficheiros .pl diferentes. Criamos então um ficheiro dedicado a toda a interface do utilizador, é lá que se encontram todos os predicados dedicados a impressão de carateres, predicados que são utilizados marioritariamente pelos predicados que se encontram no ficheiro dedicado à impresssão dos vários menus de jogo.

Uma das particularidades no desenvolvimento deste projeto foi a utilização do SWI-Prolog para a possibilidade de impressão de caracteres coloridos, uma parte bastante importante que possibilita a distinção da cor das peças.

Todas estas ações têm as suas verificações de erro, portanto nunca há um estado morto na interface, o jogador pode sempre executar uma ação.

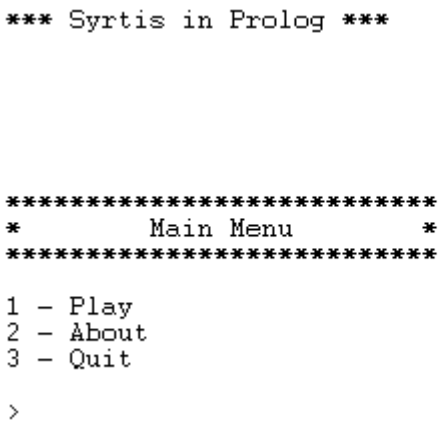

Figura 9 Menu Inicial

Na figura 9 está representado o menu inicial, o utilizador tem a possibilidade de escolher uma das três opções.

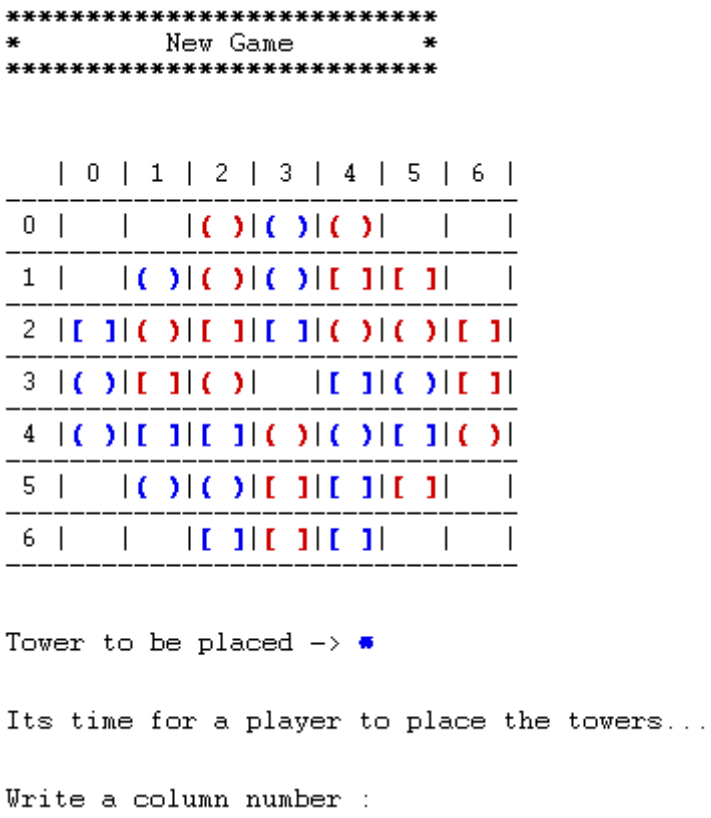

Figura 10 Escolha das posições das torres

A figura 10 representa a escolha da posição inicial das torres, com input da posição por parte do utilizador e com uma verificação de erros:

```
*****************************
*           ERROR           *
*****************************

You cant replace a tower or place a tower where there is no slot!
Pick a slot with the same shape and colour.

Press any key to continue...
|:
```

Figura 11 Verificação de erros na escolha da posição das torres

```
*****************************
*          New Game         *
*****************************

   | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-------------------------------
 0 |   |   |( )|(*)|( )|   |   |
-------------------------------
 1 |   |(*)|( )|( )|[#]|[ ]|   |
-------------------------------
 2 |[ ]|( )|[ ]|[ ]|( )|( )|[#]|
-------------------------------
 3 |( )|[ ]|( )|   |[ ]|( )|[ ]|
-------------------------------
 4 |( )|[ ]|[ ]|( )|( )|[ ]|( )|
-------------------------------
 5 |   |( )|( )|[ ]|[ ]|[ ]|   |
-------------------------------
 6 |   |   |[ ]|[ ]|[ ]|   |   |
-------------------------------

The other player should now pick a side, blue pieces play first.

1 - blue
2 - red

|: 
```

Figura 12 A escolha do lado por parte do outro jogador

Como especificado nas regras, o outro jogador deve escolher quais torres usar, como tal, também há um menu dedicado a essa parte.

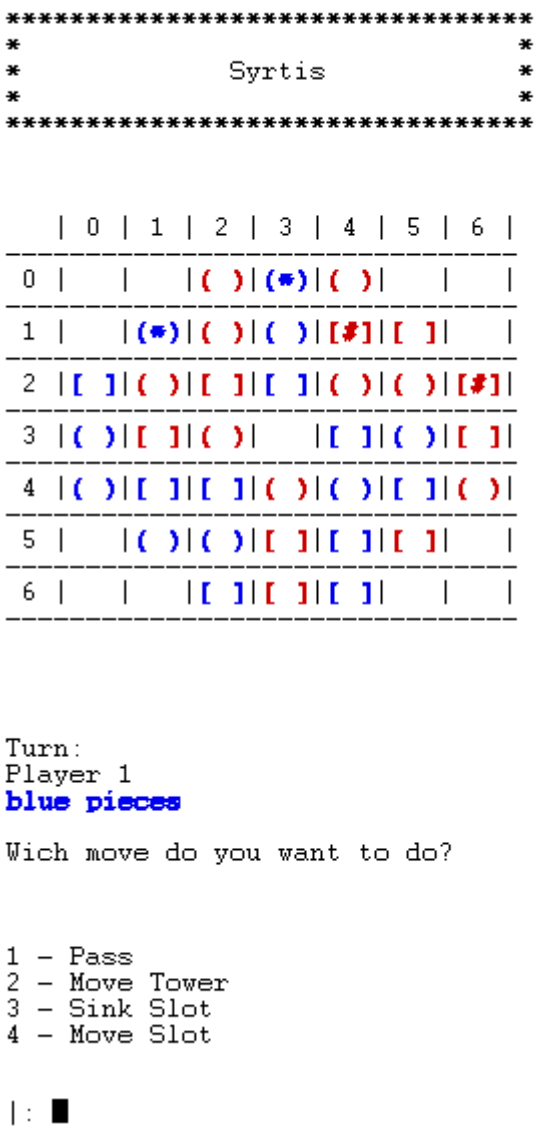

Figura 13 Menu de jogo

O menu de jogo contém várias informações sobre o jogo atual, como o número de vezes consecutivas que um jogador já passou o jogo e o número de vezes consecutivas que um jogador afundou uma casa.

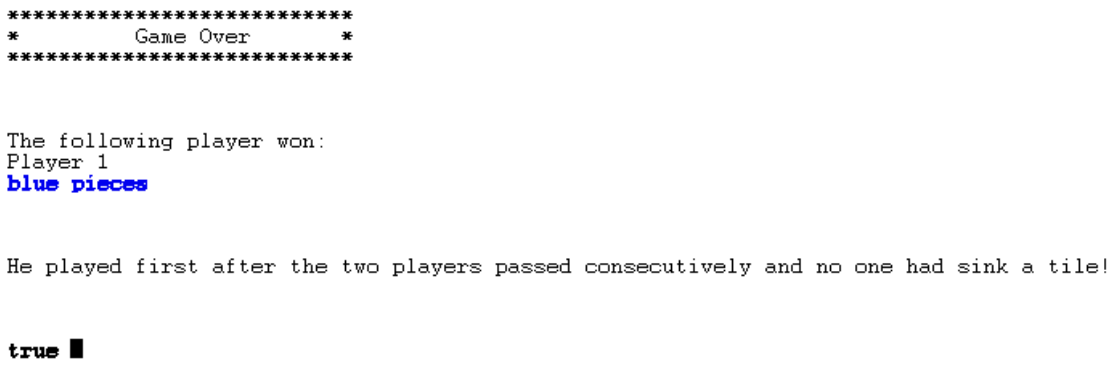

Figura 14 Fim do jogo

No fim do jogo é apresentada a informação sobre o jogador vitorioso e o motivo de vitória.

Conclusões

Para concluir o desenvolvimento deste projeto temos a acrescentar que houve muito tempo dispendido pelos dois elementos do grupo. O grupo vê um resultado final positivo, e considera que houve bastante esforço e dedicação para cumprir os objectivos propostos.

Uma das melhorias a implementar neste projecto é adicionar todos os predicados relativos ao modo de jogo "jogador vs computador".

Concluimos que é um jogo bastante complexo, um jogo que apesar de interessante tem bastantes pormenores, e por isso leva o seu tempo a ser desenvolvido.

Para finalizar, afirmamos com certeza que os nossos conhecimentos aumentaram bastante, o suficiente para perceber em que contexto poderia ser adequado usar esta linguagem de programação.

Bibliografia

- http://www.swi-prolog.org/Download.html
- http://www.swi-prolog.org/pldoc/doc_for?object=manual
- http://www.playsyrtis.com/interstate/

Anexos

Todos os anexos se encontram numa pasta ./codigo , no zip enviado.