Sixteen Stone

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 2: Sixteen Stone
Pedro Miguel Vieira da Silva – 201306032
Miguel Guilherme Perestrelo Sampaio Pereira – 201305998

11 de novembro de 2015

Resumo

O nosso projeto consistiu na implementação do jogo Sixteen Stone em Prolog. Este consiste num jogo de tabuleiro de dimensões 5x5, onde o objetivo é tirar as peças do oponente do tabuleiro, empurrando ou capturando-as.

Conseguiu-se implementar com êxito o jogo e todas as suas regras, não tendo sido possível a implementação do *bot*.

Este projeto foi produtivo, na medida em que nos permitiu aumentar os conhecimentos nesta linguagem de programação com a qual ainda não tínhamos tudo contacto.

Índice

1. Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da FEUP.

O objetivo deste projeto é implementar o jogo Sixteen Stone, recorrendo à linguagem de programação Prolog.

A maior motivação para a realização deste projeto foi a oportunidade de poder implementar um sistema de jogo tão interessante, utilizando uma nova linguagem de programação que nos permitiu aumentar os conhecimentos na área da Lógica Racional.

Este relatório serve como apoio ao projeto desenvolvido e está dividido nos seguintes pontos:

- Descrição do jogo e das suas regras;
- Lógica do jogo;
- Interface com o utilizador;
- Conclusão;
- Bibliografia;
- Anexos.

2. Sixteen Stone

2.1. História

Sixteen Stone é um jogo de tabuleiro da categoria da estratégia abstrata lançado em 2015 e criado por Gary Boyd.

2.2. Detalhes do Jogo

O jogo realiza-se num tabuleiro de dimensões 5x5, em que as casas têm cores semelhantes. Cada jogador começa com oito peças vermelhas ou azuis, colocadas alternadamente em qualquer uma das células vazias. O jogo começa assim que todas as peças tenham sido colocadas.

2.3. Objetivo

O objetivo do jogo é remover as peças do oponente do tabuleiro, empurrando-as ou capturando-as.

2.4. Regras

Começando pelo jogador vermelho, os jogadores jogam à vez fazendo diferentes ações. Os jogadores podem realizar cada uma das seguintes ações uma vez por turno, por qualquer ordem:

- Empurrar
- Movimentar
- Sacrificar

Empurrar uma peça

Os jogadores podem empurrar as peças do oponente se tiverem mais peças numa linha do que o seu oponente. Se uma peça for empurrada do tabuleiro, ela é devolvida ao conjunto de peças do oponente.

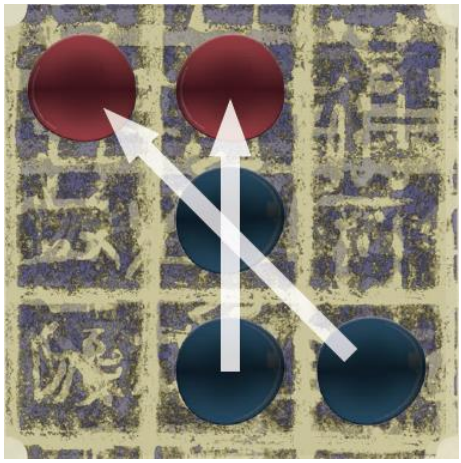

Figura 1 - As peças podem mover-se na diagonal

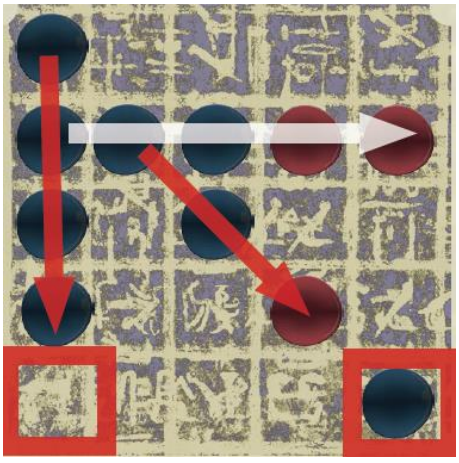

Figura 2 - 3 peças podem empurrar 2 peças. Os jogadores não podem empurrar as suas próprias peças

Movimentar uma peça

Para além de empurrar, os jogadores podem movimentar uma das suas peças para qualquer célula adjacente livre.

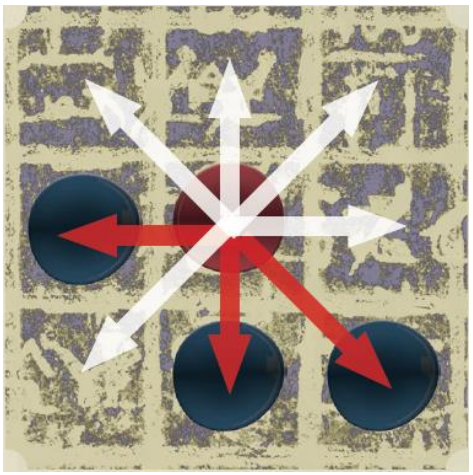

Figura 3 - Uma peça pode ser movida diagonalmente ou ortogonalmente

Capturar uma peça

Se um jogador mover uma peça para uma posição que cerque uma peça do oponente em dois lados apostos, essa peça é devolvida ao conjunto de peças do oponente e é substituída por uma peça de outra cor.

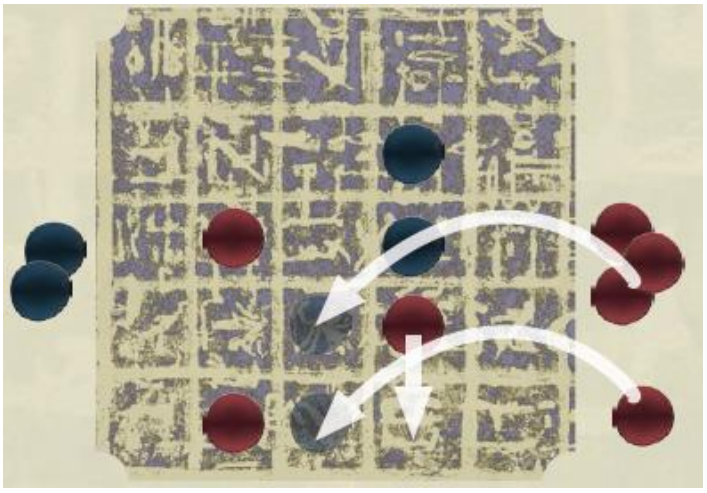

Figura 4 - Esta jogada resultará na captura de duas peças azuis, uma vez que o movimento da peça vermelha causa que ambas fiquem cercadas em lados opostos

Uma peça não é capturada se se mover voluntariamente para uma posição entre duas peças do oponente.

Sacrificar uma peça

Os jogadores podem remover permanentemente uma das peças do seu monte para fazer uma jogada adicional.

Regras adicionais e clarificações

- Os jogadores podem realizar cada uma das três ações durante a sua vez. É possível um jogador, empurrar, movimentar e sacrificar, de forma a poder empurrar ou movimentar outra vez.
- Um jogador não pode empurrar uma das próprias peças.
- As peças são empurradas apenas uma célula na direção do empurrão. Se saírem do tabuleiro, são colocadas no monte de peças.
- Enquanto se empurra, se uma das peças do jogador for movida para uma posição que não estava previamente ocupada por uma peça e cercar uma peça do oponente, o jogador pode capturar essa peça.
- Se o jogador empurrar uma peça do oponente para uma posição entre duas das peças do jogador, essa peça pode ser capturada.
- Quando uma peça do oponente é capturada, a peça com a qual essa é substituída não pode ser usada imediatamente para cercar outra peça. As peças têm de ser movidas para poderem capturar.
- Se as peças do jogador estiverem todas no tabuleiro, o jogador não pode capturar peças do oponente.

Final do jogo

Se um jogador ficar reduzido a uma única peça, o jogo acaba imediatamente e o jogador perde o jogo.

3. Lógica do Jogo

3.1. Representação do Estado de Jogo

O tabuleiro de jogo é representado por uma lista de listas de tamanho 5x5, que vai sendo alterada no decorrer do jogo com o predicado ***setMatrixElementAtWith(+Row, +Col, +Piece, +Board, -NewBoard)***. “empty” representa as células vazias, “red” as peças do jogador vermelho e “blue” as peças do jogador azul.

```
emptyBoard(Board):-
    Board = [[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 5 - Representação do estado inicial

```
middleState(Board):-
    Board = [[empty, blue, empty, empty, red],
             [empty, empty, empty, red, empty],
             [empty, red, empty, empty, blue],
             [blue, empty, empty, empty, blue],
             [empty, empty, red, empty, empty]].
```

Figura 6 - Representação de um estado intermédio

```
finalState(Board):-
    Board = [[empty, blue, empty, empty, red],
             [empty, empty, empty, empty, empty],
             [empty, empty, empty, empty, blue],
             [empty, empty, empty, empty, blue],
             [empty, empty, empty, empty, blue]].
```

Figura 7 - Representação de um estado final

3.2. Visualização do Tabuleiro

O tabuleiro é imprimido com o predicado ***printBoard(+Board)***, que recebe em ***Board*** um tabuleiro de um estado de jogo. Este predicado faz uso de outros predicados mais específicos que auxiliam na impressão do tabuleiro. As peças do jogador azul são representadas por asteriscos e as do jogador vermelho por cardinais.

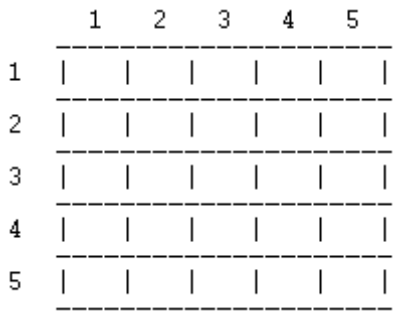

Figura 8 - Estado inicial do tabuleiro visualizado na consola

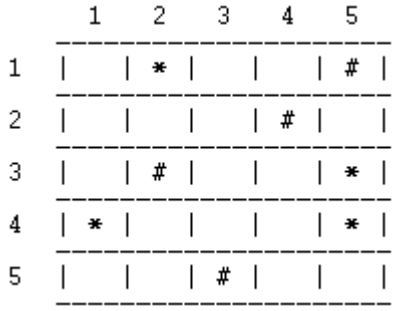

Figura 9 - Estado intermédio visualizado na consola

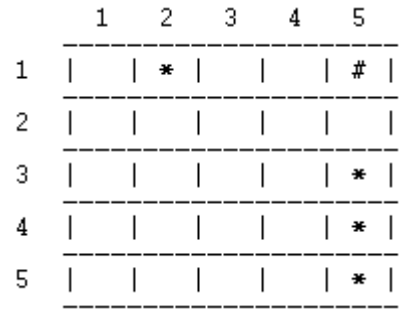

Figura 10 - Estado final visualizado na consola

3.3. Execução de Jogadas

Existem dois tipos de jogadas – movimentar e empurrar – e uma ação – sacrificar. Em cada jogada, pergunta-se o que o jogador quer fazer, executando o predicado correspondente.

- ***movePiece(+Board, -NewBoard, +Player, +BlueStones, +RedStones, -NewBlueStones, -NewRedStones)*** pede ao jogador as coordenadas da peça que ele quer mover e as coordenadas da célula para onde quer mover a peça e, depois de validados os *inputs*, executa o predicado que altera o tabuleiro;
- ***pushPiece(+Board, -NewBoard, +Player, +BlueStones, +RedStones, -NewBlueStones, -NewRedStones)*** pede ao jogador as coordenadas da peça que ele quer empurrar e a direção em que a quer empurrar. De seguida, executa o predicado ***validatePush(+Board, +Row, +Col, +Player, +DirX, +DirY)*** que verifica se o jogador tem mais peças que o seu oponente numa linha e se o jogador não está a empurrar as suas

próprias peças. Caso seja possível realizar a jogada, é executado o predicado ***pushPieces(+Board, +Row, +Col, +DirX, +DirY, -NewBoard, +Player, +BlueStones, +RedStones, -NewBlueStones, -NewRedStones)*** que efetua as alterações ao tabuleiro de jogo;

- ***sacrificePiece(+Board, +NewBoard, +Player, +BlueStones, +RedStones, -NewBlueStones, -NewRedStones)*** pergunta ao jogador que peça é que ele quer sacrificar, removendo-a do tabuleiro. Depois, pergunta qual a jogada que o jogador quer realizar, executando o predicado respetivo, de acordo com a opção escolhida.

Estes predicados utilizam, ainda, os predicados ***getCoordinates(-Row, -Col)***, que pede ao utilizador a linha e a coluna e valida se as coordenadas estão dentro dos limites do tabuleiro, e ***validatePieceOwnership(+Row, +Col, +Player, +Board)***, que verifica se a peça na célula dada pertence ao jogador que está a efetuar a jogada.

No início do jogo, é preciso colocar as peças no tabuleiro, utilizando-se o predicado ***fillBoard(+Board, +BlueStones, +RedStones, +Player)*** que pergunta aos jogadores, alternadamente, onde eles querem colocar as peças e altera o tabuleiro.

3.4. Final do Jogo

Antes de cada jogada, é verificado se o jogo chegou ao fim ou não. Utiliza-se o predicado ***checkGameOver(+Board)*** que percorre o tabuleiro e conta as peças de cada jogador. Caso algum jogador tenha apenas uma peça no tabuleiro, o jogo acaba.

4. Interface com o Utilizador

A interface com o utilizador é simples e intuitiva. No menu principal, existem cinco opções: jogador contra jogador, jogador contra computador, computador contra computador, acerca e sair. O utilizador deve introduzir o número referente à opção que quer e carregar em *Enter*.

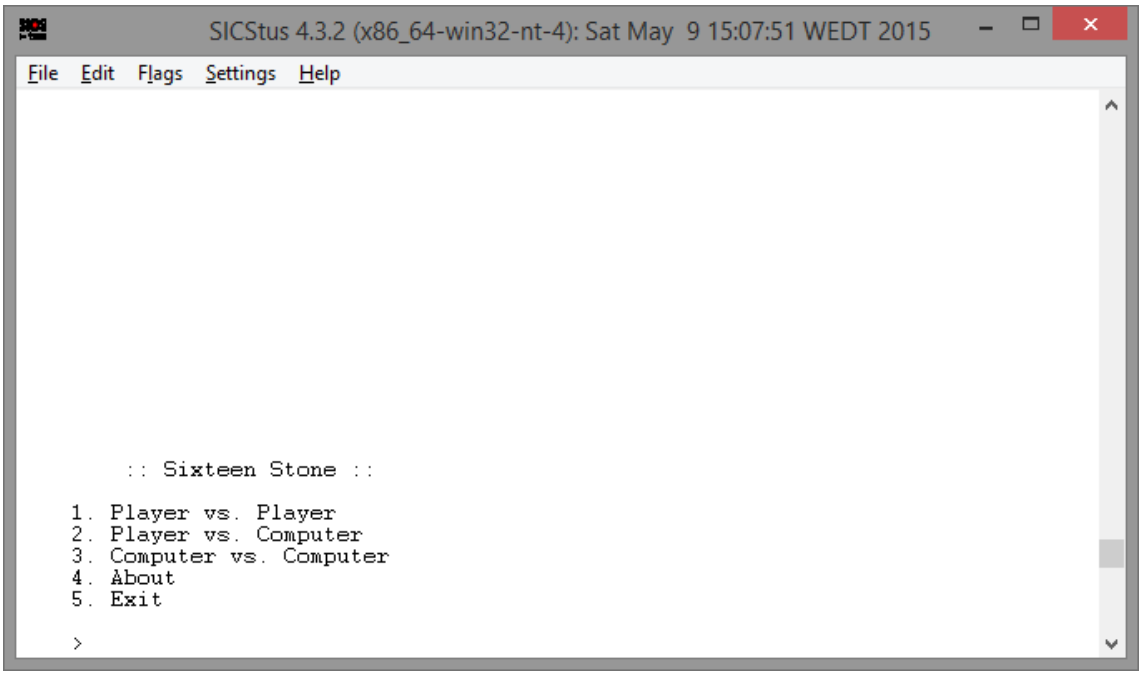

Figura 11 - Menu principal

Durante o jogo, limpa-se a consola com recurso ao predicado ***clrscr***, é imprimido o tabuleiro no estado atual e uma mensagem que indica o jogador que tem a vez e o número de peças que ele tem fora do tabuleiro. De seguida, é pedido ao jogador que indique que jogada quer realizar (ou se quer passar a vez). O jogador deve escrever o nome da jogado (não esquecendo o ponto final no fim) e pressionar *Enter*.

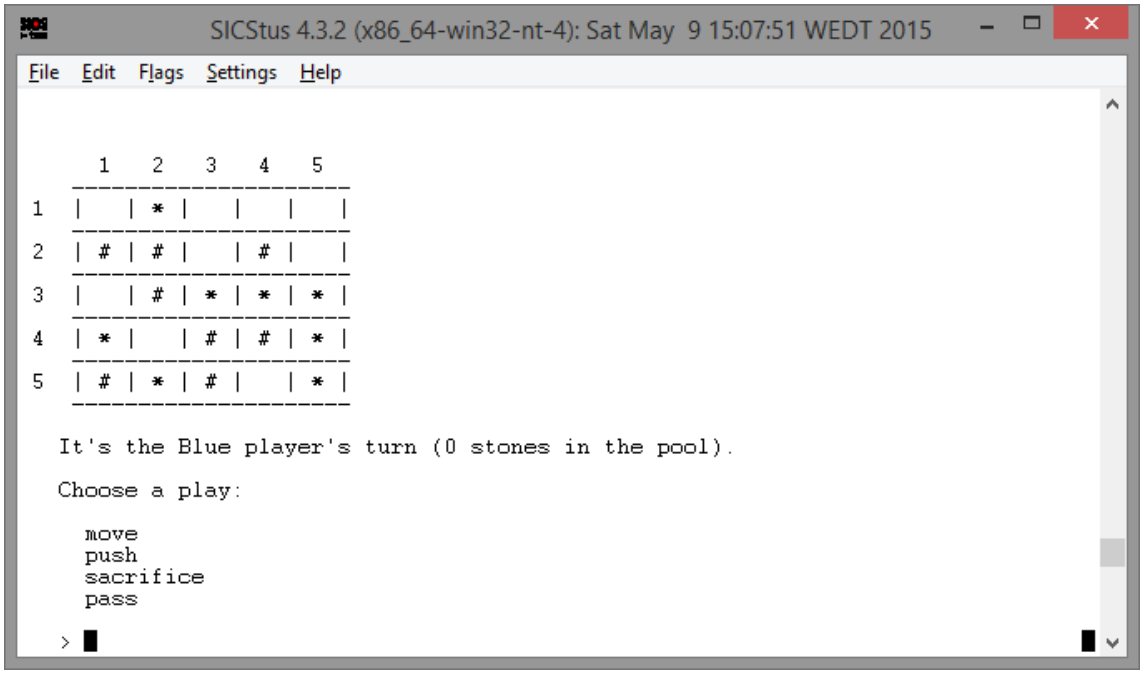

Figura 12 - Estado inicial

Caso o jogador escolha a opção de movimentar uma peça, serão pedidas as coordenadas da peça a ser movida e as coordenas de destino. Se o jogador der as coordenadas de uma peça que não seja dele ou fora dos limites do tabuleiro, é exibida uma mensagem de erro e as coordenadas são pedidas de novo. O mesmo acontece se a jogada não for válida.

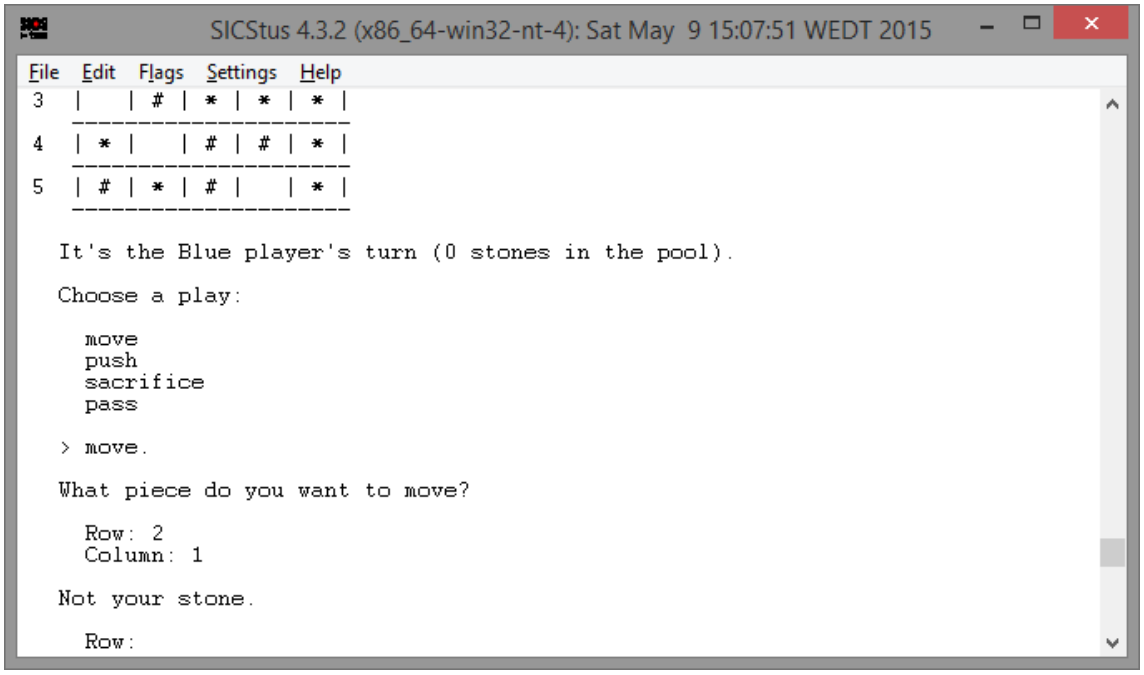

Figura 13 - Exemplo de *input* inválido

No caso do jogador escolher empurrar uma peça, são pedidas as coordenadas da peça a empurrar e a direção em que empurrar. Se o movimento não for válido, a direção continua a ser pedida até ser uma direção para onde é

possível empurrar. O jogador tem, também, a opção de sair para escolher outro tipo de jogada.

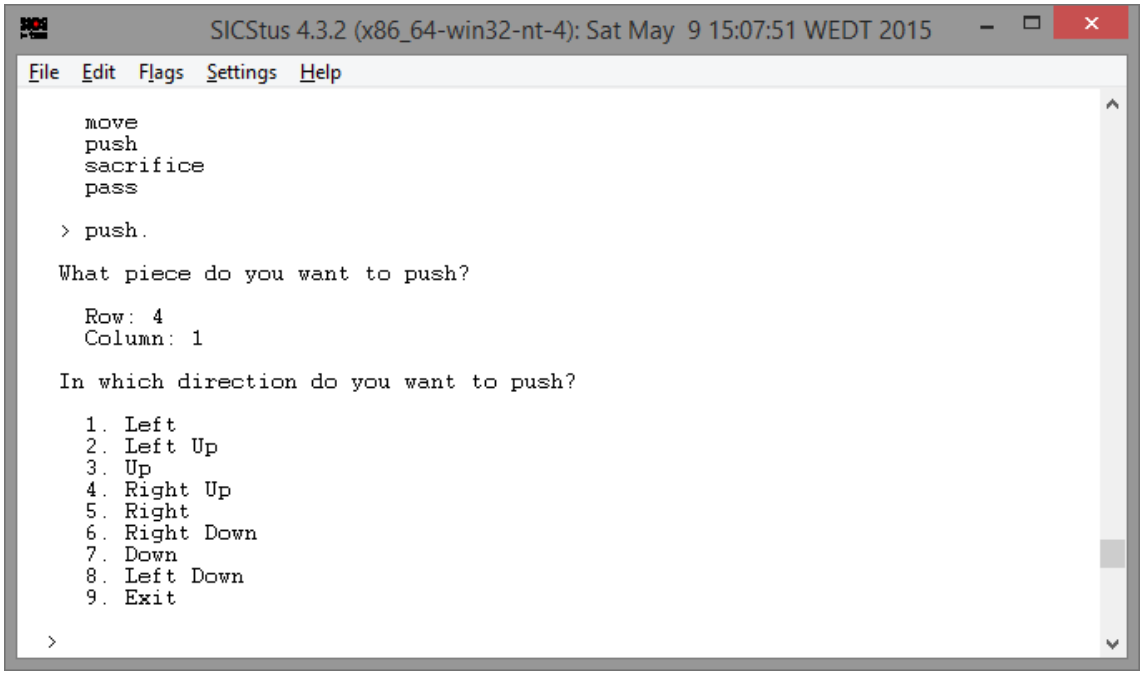

Figura 14 - Exemplo de seleção da jogada empurrar

5. Conclusões

A implementação deste jogo exigiu um grande esforço por parte do grupo, principalmente por a linguagem Prolog ser tão diferente daquilo com que estamos habituados a trabalhar.

A maior parte das dificuldades foram superadas, não tendo, no entanto, sido implementado o *bot*.

Bibliografia

"Sixteen Stone." BoardGameGeeks. N.p., 2015. Web. 2015.

Código fonte

O código fonte do projeto encontra-se na pasta *src* que está anexada junto deste relatório.