

# Sixteen Stone

## Relatório Intercalar



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo Sixteen\_Stone\_3:**  
Diogo Filipe Costa - ei11014  
Maria Teresa Chaves - up201306842

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

Novembro 2015

## Resumo

Nesta disciplina foi-nos proposto desenvolver um jogo de tabuleiro utilizando uma linguagem de programação em lógica, *Prolog*. O nosso projeto incide sobre o jogo *Sixteen Stones*. A aplicação que desenvolvemos centra-se na representação do tabuleiro de jogo quadrado, composto por células que serão ocupadas pelas pedras dos dois jogadores (Humano/Humano, Humano/Computador ou Computador/Computador) e movimentos que cada jogador poderá realizar.

Os resultados deste trabalho foram uma aplicação baseada no jogo *Sixteen Stones*, fornecendo ao utilizador uma interface em modo texto. No modo Computador/Computador utilizamos inteligência artificial de forma a que, dependendo do nível de dificuldade escolhido, as jogadas fossem o mais parecidas com as jogadas que um ser humano faria.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>O Jogo Sixteen Stones</b>	<b>4</b>
<b>3</b>	<b>Lógica do Jogo</b>	<b>6</b>
3.1	Representação do Estado do Jogo . . . . .	6
3.2	Visualização do Tabuleiro . . . . .	8
3.3	Lista de Jogadas Válidas . . . . .	8
3.4	Execução de Jogadas . . . . .	9
3.5	Avaliação do Tabuleiro . . . . .	9
3.6	Final do Jogo . . . . .	9
3.7	Jogada do Computador . . . . .	9
<b>4</b>	<b>Interface com o Utilizador</b>	<b>9</b>
<b>5</b>	<b>Conclusões</b>	<b>9</b>
<b>Bibliografia</b>		<b>10</b>
<b>A</b>	<b>Nome do Anexo</b>	<b>10</b>

# 1 Introdução

Pretende-se neste trabalho implementar, em linguagem Prolog, um jogo de tabuleiro para dois jogadores. Um jogo de tabuleiro caracteriza-se pelo tipo de tabuleiro e de peças, pelas regras de movimentação das peças (jogadas possíveis) e pelas condições de terminação do jogo com derrota, vitória ou empate. Pretende-se desenvolver uma aplicação para jogar um jogo deste tipo, usando o Prolog como linguagem de implementação. O jogo deve permitir três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador. Devem ser incluídos pelo menos dois níveis de jogo para o computador. Deve ser construída uma interface adequada com o utilizador, em modo de texto. A aplicação terá um visualizador gráfico 3D, a realizar na Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório.

## 2 O Jogo Sixteen Stones

Sixteen Stone é o nome de um jogo de tabuleiro abstrato de forma quadrada, jogado normalmente num tabuleiro de 5x5 (pode ter outras dimensões), para dois jogadores. Inicialmente, cada jogador tem 8 pedras vermelhas ou azuis (tabuleiro 5x5), colocando-as de forma alternada em qualquer uma das células livres, sendo que o jogador vermelho é o primeiro a jogar. Assim que todas as pedras estejam no tabuleiro, o jogo começa.

O jogador vermelho começa a jogar e alterna de turno com o jogador adversário. Um jogador pode no seu respetivo turno realizar cada uma das seguintes jogadas: *Push*, *Move* e *Sacrifice*. No primeiro turno de cada jogador, deve ser feito um *Push* ou um *Move*, mas nunca ambos.

### Push

Para realizar um *Push* é necessário que o jogador tenha mais pedras nessa linha que o adversário. Assim, se após um *Push*, a pedra do adversário é "empurrada" para fora do tabuleiro, então vai para o "banco" do respetivo jogador. As pedras que são "empurradas" apenas se movem uma célula na direção do *Push*, que pode ser realizado em qualquer direção. O número de pedras máximo (PE) que um jogador pode "empurrar" é igual a:

Um jogador não pode realizar um *Push* nas suas próprias pedras. Por fim, para realizar um *Push* é necessário que exista uma pedra para ser "empurrada".

$$PE = P - 1 \quad (\text{sendo } P \text{ o número das suas pedras nessa linha}) \quad (1)$$

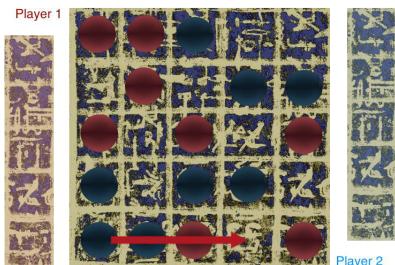


Figura 1: *Push* sem que a pedra adversária seja "empurrada" para fora do tabuleiro.



Figura 2: *Push* em que a pedra do adversário é "empurrada" para fora do tabuleiro.

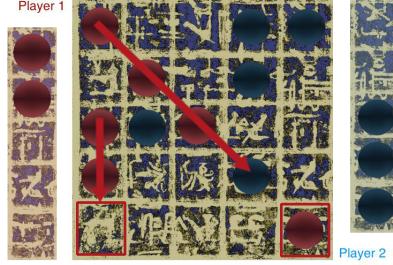


Figura 3: Push inválido: nas suas próprias pedras (a); numa célula vazia (b).

### Move

É possível realizar um *Move* para uma célula vazia em qualquer direção. Se o jogador tentar realizar um *Move* para uma célula ocupada, então é considerado uma jogada inválida.

Se um jogador realizar um *Move* e a sua pedra ficar voluntariamente cercada, esta não é capturada.

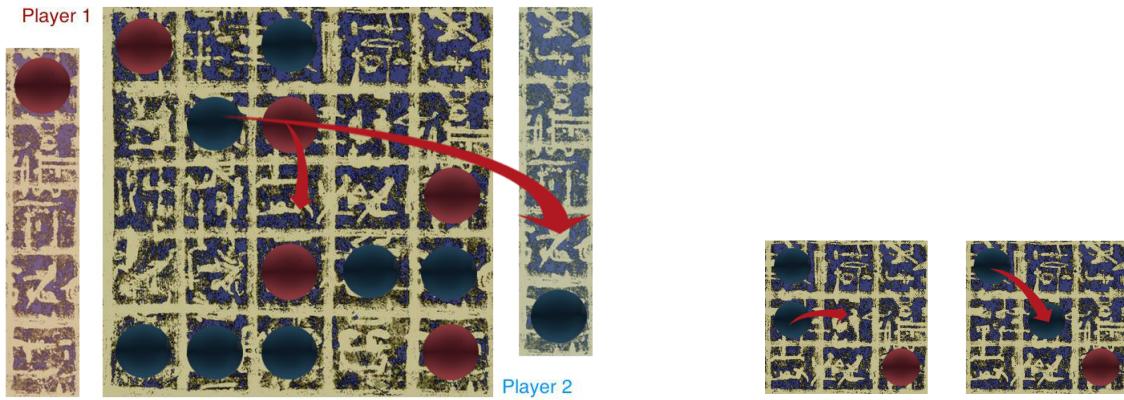


Figura 4: Pedra azul capturada após um *Move* de uma pedra vermelha.

Figura 5: *Move* válido (a) e um *Move* inválido (b).

### Sacrifice

Um jogador pode sacrificar uma pedra do seu "banco", permanentemente, para poder realizar um *Push* ou um *Move* adicional.

### Capture

Uma pedra fica cercada quando existem pelo menos em duas direções opostas pedras adversárias. Por exemplo se após um *Move* se alguma das pedras adversárias ficar cercada, então esta é capturada e substituída por uma pedra do "banco" do jogador que a capturou.

Após um *Push*, se uma das pedras do jogador atual se move para uma posição que já está ocupada por uma das suas pedras e fica a cercar uma pedra adversária então é capturada.

Se após um *Push* uma das pedras do adversário fica cercada, então é capturada.

Quando uma pedra adversária é capturada, esta pedra é colocada na *Pool* do adversário e é substituída por outra da *Pool* do jogador que efetuou a jogada.

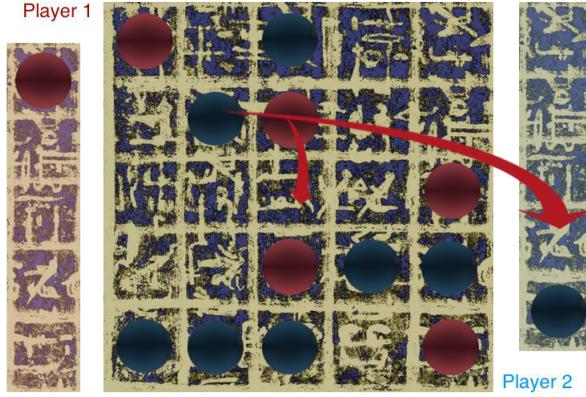


Figura 6: Pedra azul capturada após um *Move* de uma pedra vermelha.

O jogo termina quando o jogador vencedor consegue reduzir para um o número de pedras em jogo do adversário.

### 3 Lógica do Jogo

Descrever o projeto e implementação da lógica do jogo em Prolog, incluindo a forma de representação do estado do tabuleiro e sua visualização, execução de movimentos, verificação do cumprimento das regras do jogo, determinação do final do jogo e cálculo das jogadas a realizar pelo computador utilizando diversos níveis de jogo. Sugere-se a estruturação desta secção da seguinte forma:

#### 3.1 Representação do Estado do Jogo

O tabuleiro é representado por uma lista de listas  $L = \{L_1, L_2, \dots, L_n\}$ ,  $n > 0 \wedge n$  é múltiplo de 5, em que  $n$  é a dimensão do tabuleiro.

Com o predicado `make_board` é possível criar um tabuleiro (*Board*) com um tamanho *Size*. Para isso o `make_board` utiliza o predicado `make_line` para criar uma linha e depois usa o `make_board` recursivamente. De forma a tornar mais clara a visualização do tabuleiro de jogo, após criá-lo com o `make_board` utiliza-se o predicado `print_board` que o imprime no ecrã.

A figura acima demonstra o estado do tabuleiro na sua fase inicial, sem nenhuma célula preenchida. De uma forma mais visual, segue-se abaixo uma figura demonstrativa do tabuleiro.

O jogador pode realizar cada uma das três jogadas possíveis: *Push*, *Move* e *Sacrifice*. Para o *Push* é usado o predicado `push(Stone_src, Stone_dst, Dir, Board)`, onde `Stone_src` e `Stone_dst` são as coordenadas x-y (separadas por um traço) das pedras inicial e destino, `Dir` é a direção do *Push* e pode ter os valores: 0 (cima), 1 (direita), 2 (baixo) ou 3 (esquerda) e `Board` é o tabuleiro que é retornado após o *Push*.

```

| ?- Board = [[1,0,2,0,0],[0,2,1,0,0],[0,0,0,0,1],[0,0,1,2,2],[2,2,2,0,1]], draw_board(5,Board).
-----
```

0		X			
	X	0			
					0
		0	X	X	
X	X	X			0

Player 1

0	0	0	0	0				0
---	---	---	---	---	--	--	--	---

Player 2

X		X					X	
---	--	---	--	--	--	--	---	--

Figura 7: Estado inicial do jogo.



Figura 8: Tabuleiro de jogo.



Figura 9: Exemplo de um *Push*.

Para o *Move* é usado o predicado `move(Stone_src, Cell_dst, Board)`, onde `Stone_src` é a pedra que se pretende mover, `Cell_dst` é a célula de destino e o `Board` é o tabuleiro que é retornado.

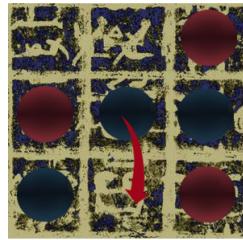


Figura 10: Exemplo de um *Move*.

Para o *Sacrifice* é usado o predicado `sacrifice(Stone, Board)`, onde `Stone` é a pedra que se pretende sacrificar e o `Board` é o tabuleiro que é retornado.



Figura 11: Exemplo de um *Sacrifice* (II).

Existe ainda um predicado `moveToPool(Stone, Player, Pool)` que move uma pedra (`Stone`) para o "banco" de um dado jogador (`Player`) retorna o "banco" do jogador (`Pool`) após a pedra ter sido movida.

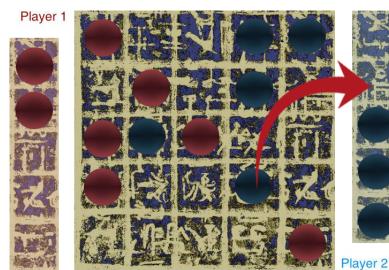


Figura 12: Exemplo de uma pedra a ser colocada no "banco" de um jogador.

### **3.2 Visualização do Tabuleiro**

(Pode ser idêntico ao descrito no relatório intercalar.)

### **3.3 Lista de Jogadas Válidas**

Obtenção de uma lista de jogadas possíveis. Exemplo: *valid\_moves(+Board, -ListOfMoves)*.

### **3.4 Execução de Jogadas**

Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Exemplo: *move(+Move, +Board, -NewBoard)*.

### **3.5 Avaliação do Tabuleiro**

Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis. Exemplo: *value(+Board, +Player, -Value)*.

### **3.6 Final do Jogo**

Verificação do fim do jogo, com identificação do vencedor. Exemplo: *game\_over(+Board, -Winner)*.

### **3.7 Jogada do Computador**

Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade. Por exemplo: *choose\_move(+Level, +Board, -Move)*.

## **4 Interface com o Utilizador**

Descrever o módulo de interface com o utilizador em modo de texto.

## **5 Conclusões**

Que conclui deste projecto? Como poderia melhorar o trabalho desenvolvido?

## A Nome do Anexo

Código Prolog implementado devidamente comentado e outros elementos úteis que não sejam essenciais ao relatório.