



UNIVERSIDADE DE ÉVORA

## Relatório Programação I

João Coutinho 58695 Miguel Aleixo 51653

Janeiro 2024

## Contents

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Implementação do Código</b>	<b>4</b>
2.1	Bibliotecas . . . . .	4
2.2	Variáveis Globais e Constantes . . . . .	4
2.3	Funções . . . . .	5
2.3.1	Campo . . . . .	5
2.3.2	Jogo . . . . .	5
2.3.3	Regras de Movimento . . . . .	6
2.3.4	Regras Complementares . . . . .	6
2.3.5	Ficheiros . . . . .	7
2.3.6	Função Main . . . . .	7
<b>3</b>	<b>Problemas Enfrentados</b>	<b>8</b>
<b>4</b>	<b>Conclusão</b>	<b>8</b>

## 1 Introdução

Ouri é um jogo de origem africana, pertencente à família de jogos *mancala*, para dois jogadores. O objetivo deste jogo é recolher mais pedras que o adversário. Todas as pedras têm o mesmo valor e vence o jogador que capturar 25 ou mais pedras.

O tabuleiro é composto por duas filas de seis buracos, aos quais damos o nome de casas, e dois buracos nos extremos designados por depósitos. Os jogadores jogam frente a frente, sendo as suas casas as que se encontram à sua frente, numeradas de 1 a 6 e sendo a casa número 1 casa mais à esquerda. Ao jogador pertence o depósito que se encontra à sua direita. No início de cada jogo são colocadas 4 pedras em cada uma das casas. O jogador que abre o jogo colhe todas as pedras de uma das suas casas e distribui as pedras, uma a uma, nos buracos seguintes no sentido anti-horário.

O projeto consistiu em implementar a lógica e regras do jogo na linguagem C dando uso a bibliotecas como **time.h** e **stdlib.h** para que o projeto fosse realizado.

Toda a estrutura e conteúdo do projeto serão apresentados nas próximas secções, descrevendo as funções implementadas no programa e como estas foram utilizadas para que fosse possível completar os objetivos que nos foram apresentados.

## 2 Implementação do Código

### 2.1 Bibliotecas

Para a realização do projeto, foram utilizadas 3 bibliotecas da linguagem C:

A biblioteca **stdio.h** para introduzir comandos básicos da linguagem como *printf()* e *scanf()* para ser possível a realização de *inputs* e *outputs* no programa, e comandos como *fprintf()* e *fscanf()* para a manipulação de ficheiros.

A biblioteca **stdlib.h** para a implementação do comando *rand()* para selecionar um número aleatório baseado numa *seed*, tornando possível a aplicação de jogadas do computador.

Por fim, a biblioteca **time.h**, a qual permite a manipulação de datas e horários e, com isso, a criação de uma *seed* baseada no segundo atual para a realização de jogadas do computador, complementando a biblioteca **stdlib.h** na implementação da pseudo-inteligência artificial do robô.

### 2.2 Variáveis Globais e Constantes

Foram definidas duas constantes com o uso do comando *define*: a constante **ROWS** = 2 e a constante **COLS** = 6, representando o número de linhas e colunas, respetivamente.

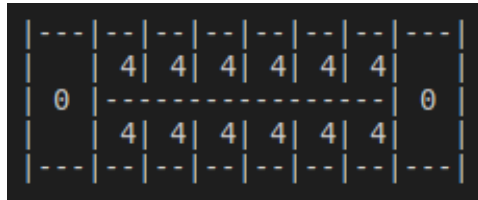
Foram utilizadas duas variáveis globais no programa, sendo elas as variáveis *board*[ROWS][COLS], que representa a matriz onde será jogado o jogo de Ouri, e as variáveis de depósitos: *depositoA* e *depositoB*, inicializadas com 0, responsáveis pelo armazenamento de pontos de ambos os jogadores. Além disso, foram ainda criadas duas variáveis *lastx* e *lasty* de modo a guardar o valor de X e Y finais após os movimentos das jogadas.

## 2.3 Funções

### 2.3.1 Campo

Estão presentes no código diversas funções essenciais para o funcionamento do jogo.

Começando pelas funções **iniciarTabuleiro()** e **printBoard()** responsáveis pela criação do tabuleiro de jogo, tendo a primeira função como argumento a matriz `board[ ][ ]` e sendo a responsável pela criação do campo onde se vai passar o jogo, colocando o número de pedras iniciais no mesmo, e a segunda responsável pela exibição do tabuleiro na consola, incluindo as variáveis *depositoA* e *depositoB* para representar os depósitos de pontos dos jogadores.

The image shows a 4x8 grid representing a game board. The grid is divided into four 2x4 quadrants by a dashed line in the middle. The top-left and bottom-right quadrants are empty. The top-right and bottom-left quadrants each contain four yellow stones, represented by the number '4'. The two cells in the middle row (row 2) are empty and contain the number '0'.

4	4	4	4	4	4	4	4
0							0
4	4	4	4	4	4	4	4

Figure 1: Representação inicial do Tabuleiro

### 2.3.2 Jogo

Estas funções apresentam regras e funcionalidades essenciais para que o jogo funcione de maneira correta.

A função **escolherModoJogo()** oferece ao jogador uma liberdade de escolha, permitindo ao utilizador, dependendo de um *input* de 1 a 2, escolher entre o modo Jogador x Jogador e Jogador x Computador.

Foi aplicada a função **checkWinner()**, que verifica após as jogadas de ambos os jogadores, se os depósitos possuem 25 ou mais pedras e, caso a função retorne o *char* de algum dos jogadores na função **main()**, a função **gameWinner()** (tomando um *char* vencedor como argumento) é chamada, exibindo o vencedor do jogo e o número de pedras que este capturou.

Por fim, foi implementada a função **trocarJogador()** que tem como objetivo a troca de jogadores após as rondas de jogo. Esta função é facilmente implementada com o uso de operadores ternários, que retorna um jogador dependendo do jogador atual.

```
// Definir jogador
char trocarJogador(char currentPlayer) {
    return (currentPlayer == jogador1) ? jogador2 : jogador1;
}
```

Figure 2: Função **trocarJogador()**

### 2.3.3 Regras de Movimento

Para os jogadores conseguirem jogar, foram implementadas uma série de funções que permitem que o movimento e capturas de peças sejam possíveis.

Para ser possível serem realizadas jogadas, são implementadas as funções **jogadareal()**, que toma como argumentos a matriz *board* e o jogador atual, e a função **jogadacomputador()** que tem como argumentos a coluna escolhida para a jogada, e o jogador atual.

Para um jogador realizar a sua jogada, este tem de introduzir um *input* entre 1 e 6, de modo a selecionar a coluna onde pretende realizar a sua jogada. Já no caso do computador, este é dotado de um *randomizer*, que seleciona um número aleatório dentro do intervalo da constante **COLS** para que o robô realize as suas jogadas.

Nas funções anteriores, estão presentes: a função **movimento()**, que permite todo o movimento das peças no tabuleiro de acordo com as regras do jogo e toma como argumentos um valor auxiliar ao seu funcionamento, as linhas e as colunas da matriz; a função **capturarPontos()** que realiza todo o sistema de captura de pontos do jogo tendo em conta as regras do jogo, verificando se as condições necessárias para que isto ocorra estão presentes após as jogadas. Esta função tem como argumentos o tabuleiro, o jogador atual, as linhas e as colunas.

### 2.3.4 Regras Complementares

Foram ainda colocadas regras complementares no programa, necessárias para que o jogo corra como esperado.

Tendo isso em conta, a função **validarJogada()**, tomando como argumentos o tabuleiro, as linhas e as colunas foi escrita com o objetivo de verificar se, caso o jogador escolha uma casa com valor igual a 1, existem na sua linha, colunas com valores superiores a 1. Caso existam, a função retorna o valor 0, invalidando a jogada, caso contrário, retorna o valor 1.

A função **linhaAdversarioVazia()** irá verificar se a linha do jogador adversário está vazia, dando uso a operadores ternários para determinar o jogador adversário e um *loop for* para verificar se alguma das colunas do adversário é diferente de 0. Caso a linha esteja totalmente vazia (todos os

valores são 0), a função retorna o valor 1. Esta função é chamada no fim das funções de jogada para que, caso após uma jogada o jogador adversário fique sem pedras no seu campo, o jogador que jogou anteriormente realiza novamente uma jogada, chamando a função da jogada de modo a colocar pedras no campo adversário. Dando uso à função anterior, a função **verificarPedraSuficiente()** é implementada com o objetivo de verificar se, caso o jogador adversário não possua pedras no seu campo, o jogador atual possui pedras suficientes para atingir a próxima linha, colocando pedras nas casas do adversário, possibilitando o funcionamento normal do jogo.

### 2.3.5 Ficheiros

Para que o jogador pudesse guardar o tabuleiro de jogo, foi criada a função **guardarTabuleiro()** que permite ao jogador, ao colocar um *input* 0 durante o jogo, insira o nome desejado para o arquivo, guardando o tabuleiro e encerrando o programa.

A função **carregarTabuleiro()** permite ao jogador abrir um tabuleiro anteriormente guardado colocando o argumento na consola. Esta função dá uso às variáveis *argc* e *argv[ ]* de modo a verificar se foi colocado um outro termo na consola, ao inicializar o programa.

### 2.3.6 Função Main

A função **main()** tem como argumentos as variáveis *int argc* e *char \*argv[ ]*. Esta, ao ser iniciada, começa por verificar o número de termos inseridos na consola ao iniciar o programa com a variável *argv[ ]* e, caso este seja igual a 2, chama a função **carregarTabuleiro()** para inserir um tabuleiro anteriormente guardado. Caso esta condição não seja verdadeira, o tabuleiro é iniciado normalmente.

Após esta verificação, é escolhido o modo de jogo e entra-se numa *loop* que é válida enquanto o seu valor for verdade. Neste *loop* é exibido o tabuleiro a cada ronda de jogo, e são chamadas as funções de jogada, dependendo do modo de jogo e jogador atual.

Sempre que as jogadas são realizadas, ocorre a verificação de vencedor que, caso seja verdadeira, ocorre um *break* que encerra o *loop* e, consequentemente, o programa.

Por fim, ocorre a troca de jogadores pela função **trocarJogador()** e o *loop* recomeça.

### 3 Problemas Enfrentados

Foram encontradas dificuldades na implementação da função **movimento()** quando era necessário as peças mudarem de linha e na função **verificarPedaSuficiente()**, que tem como objetivo forçar o jogador atual a fazer uma jogada de modo a colocar nas casas adversarias pedras quando estas estão vazias.

A função que apresentou mais dificuldade em ser implementada foi a função **capturarPontos()**, que devido a erros de lógica, recusava-se a funcionar.

Em geral as regras complementares ao jogo foram as que causaram mais problemas na sua implementação, contudo conseguimos superar as dificuldades e essas regras foram então implementadas no código, como foi pedido no enunciado.

### 4 Conclusão

Em conclusão, o trabalho está completo e de acordo com todas as regras que nos foram colocadas e o jogo encontra-se totalmente funcional, com princípio, meio e fim.

Com este trabalho, o grupo teve de realizar diversas pesquisas de modo a abranger o conhecimento relativo à linguagem C, que facilitou então a realização do projeto.

Adquirimos conhecimentos relativamente a esta linguagem e à área da Programação no geral.

Acreditamos que fizemos um bom trabalho mesmo face as dificuldades que encontrámos e esperamos que o trabalho esteja dentro das expectativas dos Professores.