Programação I: Recriando o Ouri em C

Trabalho de Projeto criado por:

Miguel Grilo	**58387**
Hugo Anacleto	**57706**
Jorge Couto	**58656**

Índice

Introdução

No âmbito da disciplina de Programação I, foi-nos proposta a criação de um programa, em C, que permitisse a alguém jogar ouri com outro jogador ou, talvez, contra o próprio computador. Este trabalho tem como objetivo testar o nosso conhecimento de programação básica, levando-nos a usar aquilo que aprendemos ao longo do semestre e, se dada a oportunidade, outros conceitos da língua de programação para desenvolvermos um modo jogador contra jogador e um modo jogador contra computador de se jogar Ouri.

Este relatório servirá de resumo geral do código feito, procurando explicar, mesmo que parcialmente, a lógica e o raciocínio do programa, além dos métodos usados para ultrapassar alguns desafios surgidos durante a fase de desenvolvimento. Adicionalmente, apesar de nos ter sido explicado quais as regras do Ouri, para melhor compreensão do relatório será feita uma explicação breve do jogo aqui.

Introdução Breve do Ouri

O Ouri é um jogo da família dos jogos mancala, sendo jogado por dois jogadores, o jogador A e o jogador B. O tabuleiro do jogo é composto por 12 casas, 2 repositórios e 48 pedras. Vence o jogador que conseguir colocar a maior quantidade de pedras no repositório, logo, 25 pedras ou mais.
Quando o jogo começa, o jogador A deve escolher uma das 6 casas do seu lado para mover as pedras. As pedras da casa que escolher serão distribuídas por todas as casas seguintes de acordo com o sentido anti-horário, uma por casa (passando à

frente do repositório). Feito o movimento, é a vez do jogador B fazer o seu movimento. Se, após o movimento, uma das pedras movidas cair em uma casa com 2 ou 3 pedras (a contar com a pedra nova), o jogador faz uma captura, colocando todas essas pedras no seu repositório. Se o jogador não tiver pedras em nenhuma das suas casas, o outro jogador é obrigado a fazer um movimento que introduza pedras nas casas daquele sem pedras. Por outro lado, se ele não conseguir fazer nenhum movimento que coloque pedras nos espaços do inimigo, então deve recolher todas as pedras para o seu repositório, acabando também com o jogo.
O Ouri tem algumas outras regras complementares. Por exemplo, se o jogo entrar em uma fase cíclica, as pedras de cada um dos lados devem ser recolhidas pelos respectivos jogadores e colocadas nos seus repositórios. Também, o jogador deve sempre escolher, nos seus movimentos, casas que tenham mais de uma pedra, podendo apenas escolher uma casa com uma pedra se não tiver nenhuma casa com mais de uma.
Apesar de simples, o problema em criar um programa que permita correr este jogo segue em desvendar a lógica por trás desse programa.

Explicação do Código

Para podermos desenvolver o código do Ouri, optamos por incluir, primeiramente, as bibliotecas *stdio.h*, *stdlib.h*, *string.h* e *time.h*. Optamos por incluir algumas bibliotecas antes de as usarmos, de modo a termos a certeza que, quando chegasse a hora de as usar, não nos fossemos esquecer de incluí-las. Entre essas bibliotecas, a única não mencionada em aula é a *time.h*, a qual foi usada para o modo de jogador vs computador.

De seguida, surgiu o problema relativo à criação dos parâmetros "principais" do jogo, como a quantidade de pedras e depósitos. Para esses, usamos um comando que não chegou a ser ensinado em aula: O #define. Com o #define podemos criar macro definições que, apesar de se comportarem como variáveis, não podem ser alteradas como as variáveis, além de se manterem por todo o código do programa.
Usando o comando #define para os parâmetros principais do jogo, de seguida fizemos a estrutura do tabuleiro, definindo-a através de um struct composto por 3 variáveis: Uma matriz do tipo int, para a inserção das casas e filas (uma fila para cada jogador) e duas variáveis int para cada um dos depósitos.
Também, focamo-nos na criação dos fatores principais do código logo pelo início. Além de definir a estrutura do tabuleiro e as macro definições, criamos a função que permite ao jogador escolher o modo de jogo jogador vs jogador (modo 1) ou jogador vs computador (modo 2), retornando a variável do modo de jogo escolhido.
A função seguinte é uma função simples, feita apenas para converter o jogador atual de valor inteiro 0 para A ou de valor inteiro 1 para B.
Como pedido no enunciado do trabalho, deveria ser dada ao jogador a possibilidade de começar o jogo através do conteúdo de um ficheiro ouri. Para isso, fizemos uma função que permite guardar o tabuleiro inicial em um ficheiro .txt, para que o jogo possa ser aberto e começado (ou recomeçado) por esse mesmo método. Naturalmente, caso o tabuleiro não possa ser guardado, demos o aviso que houve um erro ao guardar o ficheiro.
Outro problema que surgiu durante a criação do código deveu-se à validação das jogadas. Precisamos de implementar uma função que avaliasse cada um dos movimentos, impedindo o jogador de fazer jogadas impossíveis ou que fossem contra as regras do jogo. A função verifica se a jogada corresponde a uma das casas do lado do tabuleiro do jogador (1, 2, 3, 4, 5 ou 6, às

quais correspondem os valores 0, 1, 2, 3, 4 e 5 do vetor, respetivamente), se a casa está vazia e, também, se existem outras casas com mais de 1 peça.
Como se pode perceber lendo o relatório, quase todos os quesitos do programa foram feitos por função. Desse modo, reduzimos o tamanho do nosso main e, também, criamos um programa repetível e mais fácil de compreender e interagir. As capturas, as jogadas de cada jogador e do computador, a visualização (o print) do tabuleiro e outros aspetos foram feitos por função, visto que seriam partes do jogo que seriam repetidas várias vezes, ocorrendo várias vezes por partida. As jogadas de cada jogador foram feitas de modo igual, avaliando a jogada do jogador e distribuindo as pedras dependendo do movimento feito, ou realizando as capturas também.

O Computador, Função JogadaAI

Uma parte do trabalho que levou a grande confusão surgiu com o modo de “jogador vs computador”. Como seria criado um computador plausível, com o conhecimento limitado de C que temos, para o jogo? Para isso, criamos a função jogadaAI, a qual serve como nosso computador.
Apesar de envolver grande estratégia, o Ouri é um jogo simples em termos de jogadas. O jogador pode fazer até 6 jogadas diferentes, sendo limitado a menos que isso caso as regras complementares o impeçam. Por isso, com as funções complementares de validação da jogada a impedirem que qualquer um, incluindo o próprio computador, façam jogadas proibidas, implementamos um “computador” simples.
Funcionando de modo semelhante a um gerador de números, o computador escolhe um número aleatório, fazendo uma operação com esse número para retornar o resto da divisão por

6. Dependendo do número retornado, a jogada feita pelo computador é diferente. Desse modo, o jogador consegue jogar sozinho e contra um computador simples, o qual oferece chances de contra-jogada e não procura sempre a melhor jogada. O jogador não gostaria de jogar contra um computador que não oferece quaisquer chances de vitória, afinal. Por isso, ao implementar um computador "iniciante", podemos dar ao jogador chances de aprender o jogo e até mesmo vencer.

Funções Adicionais

Fora as funções já mencionadas, foram feitas algumas outras funções, como a jogada, terminar jogo e carregar. A função jogada é responsável por realizar a jogada na generalidade, informando o jogador que é o seu movimento, encarregando-se de validar os movimentos e contabilizando os movimentos das pedras feitos. A função terminar jogo determina se o jogo já terminou e qual o vencedor, verificando se a condição de vitória, definida pelo #define, foi feita com sucesso. A função terminar jogo também verifica se ocorreu um empate ou não. Por fim, a função carregar abre e lê um ficheiro de jogo, permitindo abrir um tabuleiro que tenha sido anteriormente guardado, como pedido no enunciado. Para isso, a função também faz o scan do valor dos depósitos e casas do ficheiro, permitindo continuar o jogo com as pedras nos respectivos depósitos ou casas.

O Main do Programa

O Main do programa corre todas as funções do modo pressuposto, permitindo que o jogo corra. Usando a estrutura anterior de tabuleiro, o programa cria um tabuleiro novo no main e, logo depois, cria uma string para o nome do ficheiro, caso o tabuleiro seja guardado e futuramente carregado (vale-se notar que o tamanho da string é 50, então não serão permitidos nomes de ficheiro longos).
Ainda falando do ficheiro, é o main, também, que define se o jogo é começado por ficheiro ou do 0 (do modo normal), dependendo dos argumentos usados no terminal com o início do programa. Se foram dados dois argumentos, então será usada a função de carregar tabuleiro para verificar o tabuleiro, e será dado erro se não conseguir abrir o ficheiro. Por outro lado, se foi dado apenas um argumento, então o jogo começará do modo "normal", sem ficheiro.
É o main, também, que aponta para as funções modoJogo (define se o jogo será feito em jogador vs jogador ou jogador vs computador) e printTABULEIRO, a qual imprime o tabuleiro no terminal, permitindo que o jogador veja a situação de jogo atual.
Fora isso, o main usa um "while" para deixar que as funções essenciais para que o jogo continue (jogada, print tabuleiro) se mantenham ativas. Através de um if-else-if contido dentro do while, por cada repetição o programa verifica se o jogo já terminou e, se não, verifica se o jogo está em um ciclo infinito que força a encerrar pelas regras do Ouri.
Desse modo, conseguimos elaborar um programa na língua C que permite, a quem permita acesso, jogar contra outro jogador ou contra o computador. Apesar de feito o relatório, no próprio programa foram colocados vários comentários para conferir uma explicação melhor do programa, e do seu modo de funcionamento, aos professores.