Relatório do Trabalho Prático de Programação I

Universidade de Évora 17 de janeiro de 2025.

62229 - Diana Romão Gonçalves da Silva
62124 - Eliane Nadine Fernandes Lopes
60585 - Rafael Faria Delgado

Introdução

Este relatório descreve o desenvolvimento do software "Jogo da Moeda". O jogo da moeda é um software desenvolvido com um modelamento procedural. Este modelamento é descrito principalmente na forma de fluxogramas e seu desenvolvimento foi baseado nos conceitos vistos nas aulas da disciplina. Para o desenvolvimento utilizamos a IDE VSCode, e as ferramentas GitHub e draw.io. As bibliotecas incluídas foram: stdio, stdlib, string e time.

Desenvolvimento

O desenvolvimento do software passou pelas seguintes etapas:

1. Geração dos requisitos do software.
2. Fluxograma do funcionamento do software.
3. Decisões sobre a estratégia do código
4. Validação

Requisitos

Os requisitos são descritos a seguir:

1. O software deve seguir as regras e requisitos funcionais estabelecidos na descrição da tarefa.
2. O software deve reproduzir as mesmas saídas descritas no enunciado.
3. O computador deve tentar vencer a partida se possível.

Fluxograma

O fluxograma que desenvolvemos para ilustrar o funcionamento do jogo é apresentado abaixo:

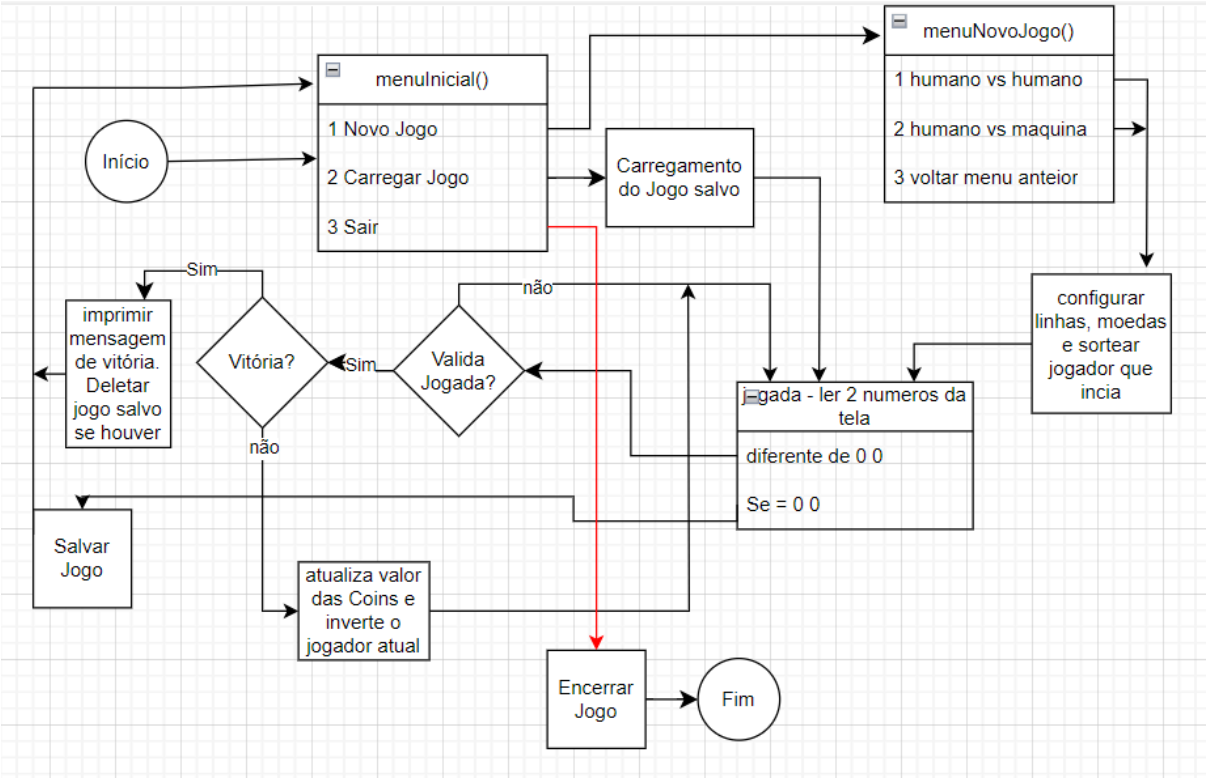

Figura 1: Fluxograma do Software Completo

Para representar “objeto” Jogo dentro do código, criamos um “struct Game” que é definido da seguinte forma com as variáveis:

-> int game_mode; que armazenado tipo de jogo, se o jogo é de Humano contra Humano ou Humano contra Máquina.

-> int current_player; um inteiro que armazena o jogador a jogar (1 sempre é o humano que iniciou o jogo, o 2 pode ser a máquina em caso de game_mode umano contra máquina, ou pode ser outro humano caso game_mode humano contra humano.

->int numColunas; armazena o número de filas do jogo, ou seja, o comprimento do *coins, com este valor fazemos o malloc de espaço para as coins.

-> int *coins; cada índice será a quantidade de moedas por fila.

-> int is_saved; um inteiro que guarda há um jogo salvo em arquivo que pode ser continuado. Quando há vitória ele retorna para 0 e o arquivo salvo é excluído.

Organizamos o código com funções para cada finalidade e estas funções chamam a próxima de forma encadeada, desta forma, a main ficou com um tamanho reduzido, mas, é perceptível o fluxo das chamadas.

A main inicia o jogo chamando a função menuInicial() apresenta o seguinte fluxograma:

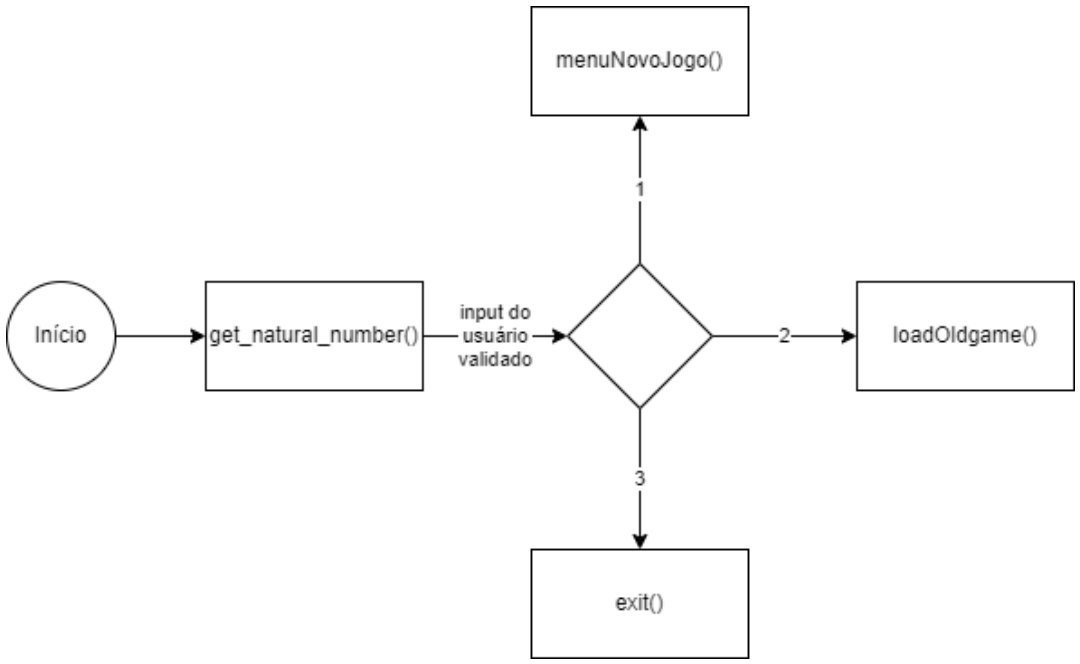

Figura 2: Fluxograma do Menu Inicial

Utilizamos a função get_natural_number() para ler dados do teclado, com ela o usuário só sai do loop se a validação confirmar que a entrada é um número natural de fato. Esta função é reaproveitada na seleção de menus, modo de jogo, número de filas e moedas por filas.

A função menuInicial() se caracteriza por repetidamente colher entradas do usuário até que satisfaça a condição de escolher 1, 2 ou 3 para direcionar a execução para a função seguinte.

A configuração do jogo se dá por meio da função menuNovoJogo() funciona de forma similar à função menuInicial() e tem como objetivo selecionar o modo de jogo preenchendo a variável game_mode do struct Game.

Ambas as opções possíveis de game_mode levam à função configCoins(). Esta função utiliza da função get_natural_number() e pergunta para o usuário, o número de filas que deseja, faz o malloc no array coins, pergunta o número de moedas para cada fila e registra estes valores nos índices/filas correspondentes. Após a configuração do jogo, o fluxo é direcionado para a função startGameHxH() que é o jogo de humano contra humano ou startGame HvM() que é o humano contra máquina.

A função loadOldgame() tem como objetivo checar se existe um arquivo salvo, e caso exista, alimenta o struct Game com as informações deste arquivo direcionando o fluxo para a função startGameHxH() ou startGameHvM() dando a devida continuidade ao jogo salvo.

Ambas as funções startGameHxH() e startGameHvM() têm estruturas similares e agrupam as etapas mostradas na Figura 1. Para iniciar o jogo em ambos os modos optamos por fazer um sorteio para saber quem seria o primeiro jogador, pois, estrategicamente quem inicia pode ter vantagem. Nestas duas funções é feita a lógica de de alternar o jogador, determinar a escolha da jogada, no caso da máquina através da função jogadaMaquinaValida(), no caso de um jogador humano, através da validação de uma entrada de jogada feita na função jogadaHumanoValida() e por fim é alterado os valores de coins do struct.

Ao jogador aqui também é dada a opção de salvar o jogo, o que leva à função saveGame(). Ao fim de cada jogada, a função checkEnd() checa a condição de vitória, e caso o jogo deva ser encerrado, o fluxo leva à função vitoria() que imprime a mensagem de vitória, remove o arquivo salvo se houver e retorna ao menu inicial.

Estratégia de Programação

Três questões foram avaliadas em relação à estratégia de programação, como tratar o fluxo dos dados entre as funções, como executar as leituras com foco nas validações destas e a quantidade de procedimentos que estariam presentes no main().

Em relação ao fluxo das informações, avaliou-se a opção de se passar o struct Game como argumento entre as funções ou usá-lo como uma variável global que pode ser acessada por todas as funções. Como o código não gera mais de uma instância do struct Game, não há um ganho ao tornar as funções capazes de avaliar diferentes instâncias do struct Game, assim, para minimizar a complexidade adotou-se o struct Game como variável global.

Em relação à leitura, preferiu-se utilizar apenas a função fgets para se evitar erros que podem ocorrer em relação ao armazenamento de buffer na memória em relação à situação onde em um mesmo código se mistura scanf e fgets onde se torna necessário utilizar funções como getchar(). A função fgets ao ler uma string inteira foi importante para a rotina de validação das entradas.

Por fim, escolheu-se a minimizar a quantidade de instruções no main() mas ainda deixando as principais rotinas do fluxogramas presentes ali.

Validação

A validação das saídas esperadas se deu por meio de testes do jogo com todos os seus modos e funções e as não conformidades foram analisadas através de mensagens de impressão de dados adicionadas em pontos relevantes do código.

Quanto ao atendimento dos requisitos da proposta do projeto:

1. O uso do tipo composto foi utilizado para caracterizar o estado e configuração do jogo. Dentro do tipo composto o array com alocação dinâmica coins foi adotado para armazenar as moedas de cada fila, com procedimentos de liberação de memória sempre que se retornava ao menu inicial para uma posterior alocação de memória em um novo jogo ou jogo salvo cumprindo assim outro requisito do projeto.
2. A manipulação de ficheiros ocorreu através das funções saveGame() e loadOldgame().
3. O uso de funções no código foi utilizado para todas as tarefas, restando ao main() apenas o encadeamento dos principais processos de forma que o entendimento e leitura do código seja facilitado. O uso de funções permitiu a reutilização destas em mais de uma parte do código. A função mais utilizada é a get_natural_number() usada em todas as chamadas por entradas do usuário. Apesar disto, o código não é 100% modular já que algumas funções modificam a variável global struct Game. Mas julgamos que isto só seria necessário caso precisássemos de mais instâncias do struct Game, como por exemplo se houvesse a necessidade avaliar tanto a jogada corrente como a anterior, se houvesse a necessidade deixar um jogo guardado na memória com a opção de voltar ao Menu Inicial e retornar ao jogo guardado, ou se houvesse a necessidade de registrar todas as jogadas feitas.
4. A validação dos dados buscou ser a mais flexível possível de modo a ser capaz de avaliar todos os cenários possíveis de inserção de dados pelo usuário e ser capaz de filtrar corretamente as entradas. Para a validação dos dados lista-se as funções jogadaHumanoValida(), jogadaMaquinaValida() e get_natural_number() onde o foco se deu em garantir que o usuário digitava uma entrada em um formato válido e que atendesse ao requisito da entrada, seja sendo apenas um número inteiro ou garantindo que existiam moedas em determinada linha para serem retiradas, ou que a linha solicitada existe.
5. Quanto à interatividade, todos os menus listados na proposta do projeto foram implementados e durante o jogo há a opção de salvar e sair digitando 0 0.

6. Às jogadas da máquina foi estabelecida uma rotina que acrescenta uma randomização às jogadas e uma mínima estratégia que busca aproveitar as chances onde se é possível vencer, no caso, quando for possível deixar o estado do jogo com apenas uma moeda.

   A rotina de randomização que busca obter o resto da divisão de um número randômico por um inteiro positivo (assim o resto "sorteado" sempre será um número inteiro entre 0 e o divisor) e usar o resto dessa divisão como valor a ser usando para escolher a linha e a quantidade de moedas a subtrair. Porém antes de entrar neste modo randômico a máquina analisa se há possibilidade de vitória, ou seja, apenas uma linha com moedas e todas as outras linhas em zero, assim, a máquina irá retirar n-1 moedas e conseguir a vitória. A máquina também analisa se está em posição de derrota, ou seja, se há apenas 1 moeda em uma fila a todas as outras filas em zero, assim ela retira esta moeda e retorna para o fluxo de vitória() do humano player1. Se nenhum destes casos se aplicar, aí sim a máquina entra em modo aleatório.

Por fim, se o jogador optar por digitar 0 0 em sua jogada, o jogo será salvo e a tela voltará para o menuInicial() onde há a opção de digitar 3 e fechar o jogo. Para salvar usamos um arquivo chamado "Cgame.txt" e as informações do struct Game que são necessárias para retomar o jogo no mesmo formato são salvas na seguinte ordem:

game_mode\n
current_player\n
numColunas\n
coins[0] coins[1] coins[2] coins[3]… coins[numColunas-1]
\n

Seguido a mesma lógica para retomar um jogo salvo caso exista os dados são fscanf na mesma ordem e já atribuídos às suas variáveis, assim como já é feito o malloc do espaço para as moedas assim que é lido o numColunas antes de ler e salvar os valores das coins.

Conclusão

O código desenvolvido cumpre os requisitos funcionais e entrega as mesmas saídas solicitadas no enunciado, buscamos organizar a lógica da forma mais simples e eficiente possível considerando o nosso nível de conhecimento e habilidade técnica nesta primeira cadeira de programação, assim como, tentamos fazer com que as jogadas da máquina não posso apenas algo aleatório que não consegue perceber uma chance de vitória.