UNIVERSIDADE FEDERAL DA GRANDE DOURADOS
Faculdade de Ciências Exatas e Tecnologia
Curso de Engenharia da Computação - FACET

BRUNO EDUARDO MARQUES GOMES

SHOW DO MILHÃO

RELATÓRIO DE PROGRAMAÇÃO

Dourados - MS
2022

1. INTRODUÇÃO E DETALHES TÉCNICOS.

O seguinte relatório tem por objetivo principal mostrar todos os detalhes técnicos e decisões tomadas na implementação de todas as funções e demais lógicas utilizadas na recriação do jogo do jogo do Show do Milhão, proposto na matéria de Laboratório de programação II.

O trabalho foi feito utilizando como base o arquivo binário perguntas.dat, onde estão todas as perguntas usadas no jogo.

No ambiente de desenvolvimento foi utilizado o aplicativo Visual Studio Code, e o compilador MingW 64 bits, mas apresenta perfeito funcionamento no Code::blocks, com compilador 32 bits.

Editor: Visual Studio Code v1.67.2

OS: Windows_NT x64 10.0.19044

Compilador: gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

2. DESENVOLVIMENTO

2.1. PERGUNTAS.

As perguntas que devem ser extraídas do arquivo binário, são lidas através da função *fread()* e armazenadas dentro da *struct* pergunta, nos campos nivel (armazena o nível da questão, sendo eles: fácil, médio, difícil e muito difícil), descrição (armazena a pergunta em si), alt (armazena as 4 alternativas) e alt_correta (armazena a alternativa correta), as questões estão iniciadas na main com a com o array "*q"* de 70 posições.

2.2. ARQUIVO.

O arquivo perguntas.dat, deve se encontrar na mesma pasta que o código, para não apresentar nenhum erro de funcionamento.

O arquivo é aberto usando a função *fopen* no modo "rb", que não altera o conteúdo do arquivo, apenas o abre em modo de leitura. Ao final da execução do programa, quando não mais necessário, é executada a função *fclose(),* que fecha o arquivo binário que foi aberto.

2.3. *struct* playerState.

Dentro dessa *struct*, que foi inicializada dentro da *main* dentro da variável "*state*", é representado o estado atual do jogador, guardando informações como o dinheiro atual, a dificuldade que o player se encontra, a quantidade de cada ajuda restante, a questão atual no jogo (1 à 16) e a questão atual do arquivo binário(1 à 70), dessa forma facilitando a manipulação e organização desses dados.

2.4. *randomNum()* - ALEATORIEDADE.

A aleatoriedade foi feita usando a função *srand()*, com a semente sendo: *time(NULL)*, da biblioteca <*time.h*>, logo no início da main. Para facilitar a geração de números aleatórios foi implementada a função *randomNum(),* do tipo inteiro que recebe como parâmetro dois números, e retorna um número aleatório entre os dois.

2.4.1. LÓGICA PARA ESCOLHA DE ALTERNATIVAS.

A lógica implementada para a escolha de alternativas nas opções onde o jogador solicita, por exemplo, ajuda da plateia ou aos universitários, é feita da seguinte forma: É gerado um número aleatório entre 1 e 10, o que significa 10% de chance para cada número cair, usando disso para selecionar a porcentagem de chance de uma alternativa ser escolhida.

2.5. CORPO DO CÓDIGO.

O corpo do código é composto pela comparação que detecta se o arquivo foi ou não aberto corretamente, caso essa informação seja validada então se inicia a leitura e armazenamento dos dados contidos no arquivo binário, e logo em seguida, é iniciado um laço de repetição *while(1)*, que só é finalizado quando o jogo recebe a função *exit(1).*

Agora dentro do laço, estamos na parte que executa cada vez que uma uma interação acontece por parte do jogador. A função *system(“cls”)*, é acionada para limpar a *HUD* do programa toda vez que o laço é reiniciado, essa funcionalidade pode não funcionar em alguns OS.

Logo em seguida é chamada a função *printQuestion()*, que basicamente imprime a questão na tela. Então o *switch()* é chamado com um parâmetro “*op”* (do tipo caractere), que pode ser uma alternativa: a, b, c ou d, e também pode ser um número para chamar as ajudas, sendo: 1, 2, 3 e 4. Para sair “op” deve valer 5, significando que o jogador desistiu do jogo. Caso o jogador insira qualquer outro valor, nada acontece.

2.6. *sameQuestion*.

A variável *sameQuestion* funciona como uma variável interruptor, que quando ativa, ou seja guarda o valor 1, a questão impressa pela função *printQuestion(),* será a mesma que foi impressa anteriormente, e quando guarda o valor 0, será atribuída ao jogador uma nova questão, toda vez que o laço principal *while(1)* se inicia *sameQuestion* recebe 0.

2.7. FUNÇÕES.

2.7.1. *printQuestion().*

A função *printQuestion()*, recebe 3 parâmetros, sendo eles a *struct* pergunta q[70], a *struct playerState *state* (por referência, pois seus valores podem ser alterados dentro da função), e o inteiro *sameQuestion.*

No início da função é feita uma comparação para saber se a questão impressa deve ser a mesma, ou se deve ser escolhida uma nova questão para imprimir na tela. Caso *sameQuestion* apresente o valor 0, de acordo com o número da questão que o jogador se encontra é atribuído um nível de dificuldade com o qual a função *selectRandomQuestion()* irá selecionar uma questão aleatória do arquivo binário para ser apresentada ao jogador.

Caso *sameQuestion* apresente valor 1, não será gerada uma nova questão, e a mesma será impressa novamente.

2.7.2. *selectRandomQuestion()*.

A função *selectRandomQuestion()* recebe como parâmetro único, a *struct playerState *state* (por referência, pois os valores podem sofrer alterações). Essa função do tipo inteiro, irá analisar a dificuldade do jogo, e retornar o número de uma questão aleatória no nível de dificuldade atual do jogador.

2.7.3. *rightQ()*.

Essa função do tipo caracter, recebe 2 parâmetros, sendo eles, a *struct* pergunta q[70] e a *struct playerState state*, e após algumas comparações, retorna a alternativa correta em relação a questão atual, em que o jogador se encontra, é usada como função auxiliar quando o jogador solicita ajuda.

2.8. AJUDAS.

2.8.1. PULAR.

Quando o jogador solicitar para pular a pergunta, é feita uma comparação para ver se ele ainda não gastou todos os pulos, caso essa comparação seja falsa, *sameQuestion* vai receber o valor 1, e portanto, a questão irá apenas repetir na tela. Em caso afirmativo, a quantidade de pulos do jogador irá diminuir em 1 unidade, e uma nova pergunta será selecionada.

2.8.2. AJUDA DA PLATEIA.

Quando o jogador solicitar a ajuda da plateia, será feito uma comparação para verificar se o jogador ainda possui esse auxílio, em caso negativo, *sameQuestion* irá receber o valor 1, e a mesma questão será impressa na tela. Em caso positivo irá iniciar um laço de repetição que dura 30 execuções, onde a questão correta, através da função *rightQ(),* será armazenada em uma variável chamada “*ans*” e será gerado um número aleatório de 1 à 10, e caso esse número for menor ou igual a 4 (representando 40% de chance para esse laço *if*) será impresso na tela entre colchetes a alternativa correta. E

consecutivamente, se o número for 5 ou 6 (representando 20% de chance) será impresso uma alternativa errada qualquer, e isso se repete para todas as alternativas. Formando assim 40% de chance da plateia escolher a alternativa correta e 20% para cada alternativa incorreta.

2.8.2. AJUDA DOS UNIVERSITÁRIOS.

Parecido com o auxílio anterior, quando solicitado será feito uma comparação para verificar se o jogador ainda o possui, e em caso negativo, *sameQuestion* recebe o valor 1 e a mesma questão será impressa na tela, em caso positivo a alternativa correta será guardada em uma variável "*ans*" e um laço *for* vai realizar 3 execuções, gerando um número aleatório de 1 à 10, se esse número for menor ou igual a 7 (representando 70% de chance para esse laço *if*) irá ser impressa na tela a questão correta entre parênteses, e caso o número aleatório seja 8, 9 ou 10, isso irá imprimir uma questão incorreta, com 10% de probabilidade para cada.

2.8.2. AJUDA DAS CARTAS.

Para esse auxílio foram utilizadas duas funções, *printCards(),* que realiza a impressão das cartas apenas, a lógica está toda contida em uma segunda função, do tipo void, chamada de *cardAux()*, essa função recebe 2 parâmetros, sendo eles, a *struct* pergunta q[70], e a *struct playerState state.* Essa função gera um número aleatório de 0 à 3, e o guarda em uma variável "*card*", e em seguida imprime esse valor para o jogador, como sendo o número de questões eliminadas.

Esse auxílio, para mostrar uma questão eliminada ao jogador, altera a letra da questão (a, b, c ou d), para a letra "x", assim mostrando que tal questão está errada. Para fazer isso, é necessário um array de 4 posições, que foi declarado globalmente (*alt*[4]), esse array vai guardar, ou a alternativa da questão ou o x, indicando que a mesma está errada, e será reiniciado sempre que *sameQuestion* for igual a 0.

Quando a função é chamada, ela entra em um laço do tipo *for* que vai ser executado de acordo com o número aleatório de questões eliminadas, definido anteriormente pela variável *card*. Dentro desse laço será feito uma comparação, para verificar se o *array alt* na i-ésima posição é diferente da alternativa correta, em caso positivo, o *array alt* na i-ésima posição irá receber "x". E então a função está quase implementada, porém, quando a letra correta não é a letra "d", ocorre um erro, pois o laço passa pela questão correta, não a alterava e o contador i recebe i++, e isso para a execução antes do tempo correto, para resolver esse problema, quando a comparação fosse negativa, ou seja, a alternativa *alt[i]* representar a opção correta, então o *else* é chamado com *card*++, para que o laço execute mais uma vez, resolvendo assim esse problema.