UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

E

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

SSC0601 & SSC0600 - Laboratório e teoria de Introdução à Ciência de Computação I

TRABALHO 5

Prof.: Adenilso da Silva Simão

Estagiários PAE: Jorge Francisco Cutigi, Leo Natan Paschoal e Samuel Lopes

Alunos:

Vinícius Santos Cubi Paulo Nº USP: 11965693

Bárbara Fernandes Madera Nº USP:11915032

São Carlos

2020

Descrição da Solução:

Para a resolução do trabalho 5, foi necessário realizar uma série de funções para os comandos do jogo Qwirkle. Além disso, ferramentas tais como ponteiros, alocações de memória, laços de repetições e make file foram usados.

Assim, a diversidade de funções presentes no programa possui como intuito organizar melhor a execução do jogo ao modo de sequenciar comandos de delimitação de tabuleiros, peças, jogadores e pontuações.

Por conseguinte, entendendo-se ponteiros como variáveis armazenadoras do endereço de memória, esses tiveram como principal funcionalidade acessar outras variáveis espalhadas pelo programa. Nesse sentido, a exemplo da ação de uma jogada, é necessário a alocação do endereço de memória para a string “peça”, copiá-la, para que, por fim, a função tabuleiro “aponte” para string da peça desejada. Com isso, a peça será redirecionada para a localização escolhida pelo jogador. Todo esse processo requer ferramentas de interligação de comandos, ou seja, requer ponteiros.

Como raciocínio análogo ao armazenamento de memória dos ponteiros, as ferramentas de alocações de memórias realizaram a função de desbloquear a memória necessária do computador para que novos ponteiros presentes do programa fossem possíveis. Caso contrário, a sequência do jogo seria interrompida pelo fenômeno “esgotamento de memória”. Logo, pela extensão do programa, é inviável a ausência nesse artificio.

Ademais, laços de repetições tais como “for” e “while” apresentam-se para delimitar condições de regras do jogo. Makefile, por sua vez, compila diferentes arquivos do programa em um só para melhorar a execução do jogo a partir de makeall e makerun.

Explicação detalhada

Detalhamentos das funções

- **Criar_tab**: Cria o tabuleiro da partida. Baseada em um limite de 108 de peças, ela é subdividida em 54 peças, sendo metade para espaços verticais e a outra metade para espaços horizontais;
- **Exp_tab:** Expande o tabuleiro para a continuação da partida. A seguinte função mede a distância da peça em relação ao fim do tabuleiro (vulgo borda). Se essa distância for menor que 1, ou seja, menor que um espaço vazio, expande-se o tabuleiro em uma casa;
- **Print_tab:** Imprime o tabuleiro no terminal após a inicialização do jogo e a cada jogada realizada. Possui como objetivo delimitar o tabuleiro em dimensões igualmente espaçadas. Realiza correções em números inválidos-números negativos ou maior que dois caracteres-, organiza o tabuleiro;

- **Del_tab:** Deleta a tabela de pontuação no caso de finalização de partida. Para isso, primeiramente, libera-se a memória alocadas no tabuleiro. Segundamente, verifica a existência de peças nas casas do tabuleiro. Desaloca colunas e linhas respectivamente. Ao final, libera todos os vetores de caracteres alocados para as peças do total da “pacote de peças” e libera o vetor armazenador de vetores;
- **Crt_pec:** Cria as peças jogáveis na partida. Aloca-se vetor de vetores de caracteres. Concatena-se as letras e os números a fim de juntá-los em uma peça. Esse procedimento ocorre 3 vezes em cada peça (uma para ler todos os chars presentes, depois para ler letras e números, por fim, para juntar letras e números).
- **Crt_jogo:** Cria a estrutura de quantidade de jogadores e o acesso de cada jogador aos seus comandos de jogada. Aloca-se memória para guardar o conjunto de caracteres presentes no nome do jogador para que esse apareça em sua vez de jogada. Vetores são alocados em 6, pois esse é o número máximo de peças que um jogador possui em cada jogada. Depois, esvazia-se as peças escolhidas, dado que essas foram deslocadas para o tabuleiro. No final da vez, novas peças são inseridas e retornam para a struct do jogador;
- **Rep_pecs:** Repassa as peças ao jogador. Baseada na aleatoriedade, ativa as peças do “pacote de peças” e as insere no conjunto de peças do jogador;
- **Dev_pecs:** Devolve as peças do jogador ao montante e retorna novas peças ao jogador. As peças pedidas para serem trocadas do jogador são verificadas no “deck” do jogador. Se existirem, elas são excluídas do arsenal do jogador;
- **Ops**: Lê a operação escrita pelo jogador e retorna um valor relativo a isso. Analisa a existência de igualdade entre número ou letra nas demais peças já existes. O comando do jogador é enviado para a função dev_pecs;
- **Jogar:** “Pega” a string e suas coordenadas ao modo de separá-las em vetor de char e em int x, y(coordenadas), respectivamente. Verifica a localização cartesiana e a peça para enviar para a função jog_valida;
- **Jog_valida:** Verifica se a posição é possível de acordo com a regras do jogo. Primeiro corrige o x e y para alinhá-los no tabuleiro alinha ao tabuleiro. Cria 3 vetores de int para registrar o que está em volta da peça. Analisa se o local desejado é um local vazio. Para o caso do início da partida, as coordenadas são x==0 e y==0, pois começa-se no meio do tabuleiro. Depois retorna “sucesso” ou “erro”, se é “sucesso” a função ops chama a outra função jogar_jog;
- **Jogar_jog:** Verifica a jogada do jogador. Corrige as coordenadas no mesmo sistema, aloca um vetor de caracteres na memória heap(malloc). Verifica se o alocamento deu certo e a existência da peça no arsenal do jogador. Se fizer parte, posiciona a peça no tabuleiro e salva a última coordenada para expandir o tabuleiro

  Atenção diferença entre as funções Jog_valida e Jogar_jog está no fato de que a primeira analisa as circunstâncias de acordo com as regras do jogo em si; a segunda, os comandos dados pelo jogador.
- **Get_int:** Função que serve para evitar entradas estranhas de char e int.
- **Main:** Função principal do programa. Nela todas as outras funções são acionadas para o bom funcionamento do Qwirkle. Ela também verifica quem terminou com a peças primeiro, ou seja, o vencedor da partida.

Funcionalidade cheat:

Tipo de jogabilidade baseada na possibilidade de escolha das peças. Dessa forma, o jogador pode digitar qualquer conjunto de dois caracteres válido. Logo, ele não está preso a arsenal aleatório de peças. Para viabilizar esse modo de jogo, foi necessário pular a etapa de verificação da existência da peça no “deck” do jogador.

Links:

-Repositório Github:

https://github.com/viniciuscubi/Qwirkle

-Vídeo da solução em grupo:

https://drive.google.com/file/d/1ukn6e4xZrW6ay1rr4K1dElSkKfs0ZeOQ/view?usp=sharing

-Vídeo de apresentação do programa em funcionamento:

https://drive.google.com/file/d/1wCKk4dm_jTo2kvD6ChyYW1PiKFOeVD-E/view?usp=sharing