Trabalho final Algoritmos e Programação

Vítor Feijó e Gabriel Nunes

INF01202

Como jogar?

O jogo desenvolvido é inspirado no clássico Jetpack Joyride. O personagem, representado por um retângulo, precisa navegar por um mapa utilizando a tecla de espaço ou o botão esquerdo do mouse para impulsionar-se para cima. O objetivo do jogador é coletar o maior número possível de moedas enquanto avança pelo mapa, desviando de obstáculos, os espinhos. Caso o personagem colida com um espinho, o jogo é finalizado. A dinâmica do jogo foca na habilidade do jogador em evitar colisões e maximizar a coleta de moedas na maior distância possível

FUNCIONALIDADES E IMPLEMENTAÇÃO DO JOGO

Mapa do jogo: Foram criadas duas funções para tratar da parte do mapa do jogo, uma para a leitura do mesmo e outra para a subdivisão deste.

Leitura do mapa: A função leMapa recebe como parâmetros uma matriz de caracteres e um vetor de caracteres, sendo responsável por preencher a matriz com os dados de um arquivo de texto. Primeiramente, o arquivo é aberto para leitura. Em seguida, um laço while percorre o arquivo caracter por caracter até o final (EOF), verificando se cada caractere é válido (como 'C', 'X', ' ' ou 'Z'). À medida que os caracteres são lidos, eles são armazenados na matriz que representará o mapa. Quando o preenchimento é concluído sem erros, a função retorna 0, indicando que o processo ocorreu corretamente.

```
int leMapa(char matriz[][COLUNAS], char nome[STRTAMANHO]) {

  //Abre o arquivo
  FILE *arq = fopen(nome, "r");

  //Testa a abertura correta do arquivo
  if (arq == NULL) {
    printf("Erro/n");
    return 1;
  }

  //Definições para a trasposição da matriz
  char parte;
  int linha = 0;
  int coluna = 0;
```

```
  while ((parte = fgetc(arq)) != EOF){  //Testa se o final do arquivo foi chego
                                        //junto da atribuição do caracteres no char parte

    if (parte == '\n') {
      // Ignora quebras de linha
    }

    if (parte == 'C' || parte == 'X' || parte == ' ' || parte == 'Z') {
      // Preenche a matriz com os caracteres válidos
      matriz[linha][coluna] = parte;
      coluna++;
      if (coluna == COLUNAS) {
        coluna = 0;
        linha++;  // Avança para a próxima linha quando a coluna atingir 240
      }
    }
  }

  //Fecha o arquivo
  fclose(arq);

  return 0;
}
```

Subdivisão do mapa: o objetivo da subdivisão do mapa é cumprir com o requisito " O jogo utiliza seções dessa matriz, de tamanho 12×30, que são exibidas em um determinado momento. À medida que o tempo de jogo passa, deve-se selecionar uma partição aleatória do tamanho desta seção a partir da

matriz lida do arquivo correspondente aquela fase". Portanto, foram utilizadas duas funções, uma para subdividir o mapa, e outra para escolher aleatoriamente uma partição do mesmo. A função "PegaSubdivisao1Mapa" recebe a matriz onde armazenamos a leitura do mapa e outra matriz de tamanho menor onde será armazenado o primeiro submapa de tamanho 12 x 30. A extração do primeiro submapa é realizada utilizando um laço for que navega pelas linhas e colunas da matriz original e as armazena no submapa. Para cada posição [i][j] da matriz submapa, o valor correspondente da matriz original mapa é copiado para essa posição, utilizando o cálculo do índice de coluna com a expressão (numero * COLUNAS_MATRIZ_MENOR) + j.

```
void PegaSubdivisao1Mapa(char submapa[][COLUNAS_MATRIZ_MENOR], char mapa[][COLUNAS]){
  int numero = 0;
  int i,j;

  //Pega a primeira submatriz 12x30 pelo numero ser sempre 0
  for(i = 0;i < LINHAS;i++){
    for(j = 0;j < COLUNAS_MATRIZ_MENOR;j++){
      submapa[i][j] = mapa[i][(numero*COLUNAS_MATRIZ_MENOR) + j];
    }
  }

}
```

Escolha de partições aleatórias: A função PegaSubdivisaoAleatoriaMapa tem como objetivo selecionar aleatoriamente uma das 8 subdivisões do mapa para ser usada no jogo. Para isso, ela gera um número aleatório entre 0 e 7, que determina qual submapa será extraído. O processo de extração é realizado utilizando um laço for similar ao da função PegaSubdivisao1Mapa, percorrendo as linhas e colunas do submapa e preenchendo-o com os valores correspondentes da matriz original, mapa.

```
void PegaSubdivisaoAleatoriaMapa(char submapa[][COLUNAS_MATRIZ_MENOR], char mapa[][COLUNAS]){
  int numero;
  int i,j;

  //Pega um numero aleatorio entre 0 e 7
  numero = (rand() % NUM_SUBMAPAS);

  //Usa o numero aleatorio para escolher uma das 8 submatrizes para ser o mapa do jogo
  for(i = 0;i < LINHAS;i++){
    for(j = 0;j < COLUNAS_MATRIZ_MENOR;j++){
      submapa[i][j] = mapa[i][(numero*COLUNAS_MATRIZ_MENOR) + j]; //j é somado ao numero aleatorio vezes o
                                                                  //numero de colunas da matriz menor
    }
  }
```

Mapa infinito: Com o objetivo do jogo dar a impressão de movimentação do personagem ao longo do mapa no eixo **x**, foi preciso implementar a movimentação do mapa de forma infinita, uma vez que o jogador deve permanecer parado em uma coluna. Para isso, a solução adotada foi a utilização de duas seções (ou submapas) do mapa que se deslocam continuamente na direção negativa do eixo **x**. Quando uma seção sai completamente da tela, ela é reposicionada ao lado da outra, criando a ilusão de um ambiente infinito.

A cada atualização do jogo, as seções se movem para a esquerda com uma velocidade determinada. Quando uma seção alcança o limite de deslocamento (ou seja, sai da tela), ela é reposicionada no ponto inicial e recebe uma nova parte do mapa, gerada aleatoriamente. Esse processo garante que o jogo nunca tenha um "fim" no cenário, proporcionando ao jogador a sensação de mapa infinito.

```
//deslocamento
deslocamentoX1 -= velocidade;
deslocamentoX2 -= velocidade;

if (deslocamentoX1 <= -COLUNAS_MATRIZ_MENOR * TAMANHO_BLOCO) {
    deslocamentoX1 = COLUNAS_MATRIZ_MENOR * TAMANHO_BLOCO;
    PegaSubdivisaoAleatoriaMapa(submapa1, matrizMapa);
}

if (deslocamentoX2 <= -COLUNAS_MATRIZ_MENOR * TAMANHO_BLOCO) {
    deslocamentoX2 = COLUNAS_MATRIZ_MENOR * TAMANHO_BLOCO;
    PegaSubdivisaoAleatoriaMapa(submapa2, matrizMapa);
}
```

Personagem do Jogo e Movimentação:

Duas funções foram desenvolvidas para interagir de forma dinâmica e implementar a jogabilidade do projeto. Primeiramente, o personagem do jogo foi representado por um retângulo de cor preta, com dimensões 40 x 40 pixels. Sua posição inicial foi definida pelas variáveis jetpack_x e jetpack_y, correspondendo ao eixo X e Y, respectivamente. A variável jetpack_x foi inicializada com o valor 150, e jetpack_y com o valor 225, posicionando o personagem no mapa.

Além disso, foi criada uma variável chamada velocidade, que controla a movimentação do personagem no eixo Y, permitindo que ele se desloque verticalmente. Esse controle de velocidade atende ao requisito de que "o jogador sempre se mantém em uma determinada coluna de uma seção carregada do mapa", garantindo que a movimentação do personagem ocorra apenas ao longo do eixo vertical, sem afetar sua posição horizontal.

```
int jetpack_x = POSICAO_X_JOGADOR;              // posicao x do personagem fixa na tela
int jetpack_x_mapa = POSICAO_X_JOGADOR;           // posicao x do personagem no mapa
int jetpack_y = ALTURA / 2;      // posicao y do personagem
float jet_speed_y = 200;
int dy = 0;
```

A função “Movimentacao” recebe três parâmetros: um ponteiro para a posição vertical do personagem (y), um ponteiro para a sua velocidade de movimentação (jet_speed_y) e um ponteiro para indicar a direção da intenção de movimento (se o personagem deve se mover para cima ou para baixo).

Se a tecla de espaço ou o botão esquerdo do mouse estiverem sendo pressionados, isso indica que o personagem deseja se deslocar para cima. No entanto, para garantir que o personagem não ultrapasse os limites da tela (com altura máxima de 450), é realizada uma verificação utilizando a função “deve_mover”. Essa função controla a movimentação do personagem, permitindo ou não o movimento dependendo dos limites superior e inferior da tela.

A lógica assegura que o personagem se mova para cima quando desejado, mas sem ultrapassar os limites da área visível do jogo.

```
int deve_Mover(int dy, int jetpack_y){

    if((dy == 1 && jetpack_y <= LIM_SUPERIOR) || (dy == -1 && jetpack_y >= LIM_INFERIOR)){
        return 0; // jetpack não pode se mover
    }else{
        return 1; // jetpack pode se mover
    }

    return 1;
}
```

Quando a tecla de espaço ou o botão esquerdo do mouse não estão sendo pressionados, é necessário simular a gravidade que puxa o personagem para baixo. Para isso, foi implementada uma aceleração negativa na direção vertical, aplicando uma velocidade no sentido descendente. Essa velocidade negativa representa a força da gravidade, fazendo com que o personagem caia de volta ao solo quando o impulso para cima não é ativado.

```
void Movimentacao(int *jetpack_y, float *jet_speed_y, int *dy){

  if(IsKeyDown(KEY_SPACE) || IsKeyDown(MOUSE_BUTTON_LEFT)){
    *dy = 1;
    if(deve_Mover(*dy,* jetpack_y)){
      *jetpack_y -= *jet_speed_y * GetFrameTime();    // aumenta ou diminui a distância que se move o personagem em funcao do fps
      WaitTime(SECONDS);
    }
    }
    else if(IsKeyUp(KEY_SPACE) || IsKeyUp(MOUSE_BUTTON_LEFT)){
      if(*jetpack_y < LIM_INFERIOR)
      *jetpack_y += GRAVIDADE * GetFrameTime();  // aplicar a gravidade se a tecla para cima nao esta sendo pressionada
    }

}
```

Desenho do Mapa e Colisões:

Para preencher a interface gráfica do jogo, foi criada uma função que desenha os retângulos correspondentes aos elementos do mapa. Simultaneamente, essa função realiza a verificação de colisões entre o personagem do jogo e os blocos de colisão, representados pela letra ‘Z’, assim como as moedas,

representadas pela letra ‘C’. Dessa forma, a função renderiza o mapa e também permite a interação entre o personagem e os objetos do jogo.

Além disso, ao detectar a colisão com uma moeda, a função atualiza a pontuação do jogador, incrementando o número de moedas coletadas. No caso das colisões com os blocos de colisão (representados pela letra ‘Z’), a função verifica se o personagem entrou em contato com algum desses obstáculos, permitindo que quando o personagem choque com o bloco “Z”, o jogo acabe.

Esse processo de renderização e verificação de colisões acontece ao mesmo tempo em que os blocos são desenhados, garantindo que o mapa esteja sempre visível para o jogador enquanto ele interage com os elementos do jogo.

```
int DesenhaRetanguloeColisao(int x, int y, int jetpack_x, int jetpack_y, char matriz[LINHAS][COLUNAS_MATRIZ_MENOR], int *moedas, Jogador jogadores[])
  int i,j;
  //Variaveis de colisão
  int colisao = 0;
  int colisaoMoeda = 0;
  int colisaoBomba = 0;

  for (i = 0; i < LINHAS; i++) {
    for (j = 0; j < COLUNAS_MATRIZ_MENOR; j++) {

      //Condições de colisão
      colisaoMoeda =
        jetpack_x >= (x + j * TAMANHO_BLOCO) &&
        jetpack_x <= (x + j * TAMANHO_BLOCO + TAMANHO_BLOCO + 10) &&
        jetpack_y >= (y + i * TAMANHO_BLOCO - 39) &&
        jetpack_y <= (y + i * TAMANHO_BLOCO + TAMANHO_BLOCO);

      colisaoBomba =
        jetpack_x >= (x + j * TAMANHO_BLOCO + 30) &&
        jetpack_x <= (x + j * TAMANHO_BLOCO + TAMANHO_BLOCO) &&
        jetpack_y >= (y + i * TAMANHO_BLOCO - 30) &&
        jetpack_y <= (y + i * TAMANHO_BLOCO + TAMANHO_BLOCO);

      Color squareColor = CorDoChar(matriz[i][j]); //Analisa o char da matriz e determina a cor do quadrado
```

```
      DrawRectangle(x + j * TAMANHO_BLOCO, y + i * TAMANHO_BLOCO, TAMANHO_BLOCO, TAMANHO_BLOCO, squareColor);  //Desenha o quadrado

      //Teste das colisões
      if(colisaoMoeda){
        if(matriz[i][j] == 'C'){
          matriz[i][j] = ' ';
          *moedas += 1; //Soma uma a quantidade de moedas
          jogadores[3].pontuacao += 1; //Soma um a pontuação do jogador
        }
      }
      if(colisaoBomba){
        if(matriz[i][j] == 'Z'){
          //matriz[i][j] = ' ';
          colisao = 1;
        }
      }


    }
  }
  return colisao;
}
```

Menu: O menu do jogo é implementado por meio da função DesenhaMenu, que recebe como parâmetros dois botões, botaoJogo e botaoLeaderboard, além de um ponteiro para a variável GameScreen, que controla a tela atual do jogo. A função tem como objetivo exibir o menu principal, onde o jogador pode interagir com os botões para acessar diferentes partes do jogo.

Dentro da função, é feita a verificação de interação com o mouse sobre os botões. Se o jogador posicionar o cursor sobre um botão, a cor deste botão muda para indicar que ele está ativo. Quando o jogador clica em um dos botões, a variável GameScreen é alterada para a tela correspondente, como GAMEPLAY ou LEADERBOARD, permitindo que o jogo avance para a seção desejada.

Além disso, o menu também exibe instruções ao jogador, como pressionar as teclas N ou L para acessar o jogo ou o leaderboard, respectivamente, facilitando a navegação pelo jogo.

```
void DesenhaMenu(Botao botaoJogo, Botao botaoLeaderboard, int *GameScreen) {

  if(MouseNoBotao(botaoJogo)) {
    botaoJogo.cor = PINK;
  } else {
    botaoJogo.cor = RED;
  }

  if(MouseNoBotao(botaoJogo) && IsMouseButtonPressed(MOUSE_BUTTON_LEFT)){
    *GameScreen = GAMEPLAY;
  }

  if(MouseNoBotao(botaoLeaderboard)) {
    botaoLeaderboard.cor = PINK;
  } else {
    botaoLeaderboard.cor = RED;
  }

  if(MouseNoBotao(botaoLeaderboard) && IsMouseButtonPressed(MOUSE_BUTTON_LEFT))
    *GameScreen = LEADERBOARD;
  }
```

Leaderboard:

O Leaderboard é implementado utilizando duas funções: DesenhaLeaderboard e GravaLeaderBoard.

A função “DesenhaLeaderBoard” recebe como parâmetros um ponteiro para a struct Jogador, o botao do menu e um ponteiro para a tela do jogo. Seu objetivo é desenhar na tela a colocação dos usuários ao terminar o jogo. Portanto, são criados 3 vetores de char para armazenar as 3 melhores pontuações gerais. Se a pontuação do novo jogador é maior que a pontuação de qualquer um dos jogadores anteriores, o novo jogador é colocado no ranking dos melhores jogadores. Para essa funcionalidade ser implementada, foi utilizado uma sequência de “ifs” que ordena os jogadores conforme a pontuação e os substitui ou não.

A função GravaLeaderBoard recebe como parâmetros um ponteiro para a estrutura jogadores, que contém as informações dos jogadores, e realiza a leitura e escrita de dados em um arquivo binário. Inicialmente, a função abre o arquivo em modo binário para permitir tanto a leitura quanto a escrita dos dados. Em seguida, um laço for é utilizado para iterar sobre os jogadores, preenchendo o campo destinado ao nome de cada jogador na estrutura, garantindo que as informações sejam corretamente salvas no arquivo e possam ser recuperadas posteriormente para exibição no leaderboard.

```
int GravaLeaderboard(Jogador *jogadores) {
//Pega o arquivo binario com o top 3 jogadores e grava em uma estrutura dada

//4 bytes de int para a pontuação e depois mais 30 bytes de nome do cara(3 jogadores)
  FILE *arq = fopen("leaderboard.bin", "rb+");

  if(arq == NULL){
    printf("leitura incorreta do arquivo");
    return 1;
  }

  int i;

  for (i = 0; i < NUMJOGADORES; i++) {
    fread(&jogadores[i], sizeof(Jogador), 1, arq);
  }

  fclose(arq);
  return 0;
}
```

```
int GravaNovoJogador(Jogador jogadores[]){

  FILE *arq = fopen("leaderboard.bin", "wb");

  if(arq == NULL){
    printf("leitura incorreta do arquivo");
    return 1;
  }

  if(jogadores[3].pontuacao >= jogadores[0].pontuacao){
    fwrite(&jogadores[3], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[0], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[1], sizeof(Jogador), 1, arq);
  }
  else if(jogadores[3].pontuacao >= jogadores[1].pontuacao){
    fwrite(&jogadores[0], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[3], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[1], sizeof(Jogador), 1, arq);
  }
  else if(jogadores[3].pontuacao >= jogadores[2].pontuacao){
    fwrite(&jogadores[0], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[1], sizeof(Jogador), 1, arq);
    fwrite(&jogadores[3], sizeof(Jogador), 1, arq);
  }
```

Estruturas utilizadas no jogo:

A estrutura Jogador é utilizada para guardar o nome dos jogadores após o gameplay, uma vez que guarda a pontuação e cria um vetor de char para esse fim.

A estrutura “Botao” é utilizada para conter os parâmetros dos botões criados nos menu, tais como a posição, tamanho e a cor.

```
typedef struct {
    float pontuacao;
    char nome[STRTAMANHO];
} Jogador;

typedef struct {
  Rectangle ret;
  Color cor;
} Botao;
```