Entrega do Trabalho Final: MineFake

Thiago Vito Paes de Farias (00342013), Fernando Kaspary Fink (00342166)

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

thiagovito2468@gmail.com
fernandokfink@gmail.com

Resumo. *A esquematização da organização do trabalho foi dividida entre as seguintes tarefas: o integrante da dupla Thiago responsável maioria do código front-end. Já o integrante Fernando, pela maioria do back-end. Houve extrema participação de ambas as partes em todas as funções.*

1. Enredo

O trabalho final foi baseado no jogo Minecraft, pois seus assets já estavam à disposição na internet.

2. Arquitetura da aplicação desenvolvida (módulos, estruturas e funções utilizadas)

No projeto, foram utilizados 10 módulos, 5 estruturas e x funções.

2.1 Módulos:

animação.h: animação do jogador

coordenadas.h: localização do jogador

item.h: itens do jogador

jogador.h: informações e parâmetros do jogador

mapa.h: informações do mapa

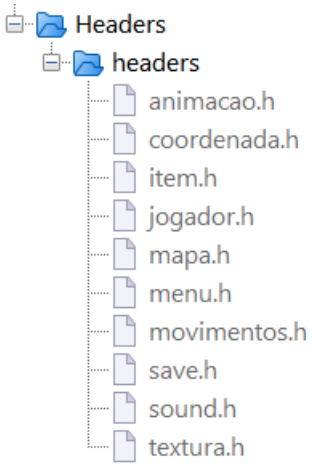
menu.h: informações do menu

movimentos.h: movimentação do jogador

save.h: salvamento do jogo

sound.h: sons do jogo

textura.h: textura do jogo



2.2 Estruturas:

Mapa (utilizada em tarefas práticas):

```
typedef struct {
    char mapa[MAPA_L][MAPA_C];
    Localizacao dimensao;
    int fase, baus, bausAbertos;
    char porta;
    Jogador jogador;
} Mapa;
```

// Função: informar parâmetros como o mapa, a localização do jogador, qual fase o jogador está, os baús que o jogador abriu, se ele está sobre uma porta e informações da estrutura Jogador (nome, pontos etc.).

Jogador (utilizada em tarefas práticas)

```
typedef struct {
    char nome[TAM_NOME];
    Localizacao posicao;
    int vidas, pontos, skin;
    bool temChave, facingLeft, temItemFase, temArmadura;
} Jogador;
```

// Função: informar parâmetros como nome do jogador, localização, vidas, pontos, skin escolhida, se o jogador já obteve a chave para passar de fase, informações de animação como facingLeft, e se o jogador tem armadura (proteção contra explosão).

Texturas (Texture2D foi tirado da Raylib):

```
typedef struct {
    Texture2D blocos[NUM_BLOCOS];
    Texture2D player[SKINS][NUM_TEXTURAS_SKIN];
    Texture2D background;
    Texture2D interface[NUM_TEXTURAS_GERAL];
} Texturas;
```

// Função: adicionar texturas ao jogo com a função pronta da Raylib “Texture2D”.

Item:

```
typedef struct {
    char nome[TAM_NOME_ITEM], descricao[TAM_DESC];
    int tipo, ID;
} Item;
```

// Função: informar o nome do item, a descrição, e o código do item.

Localizacao (utilizada em tarefas práticas):

```c
typedef struct {
    int linha, coluna;
} Localizacao;

Localizacao criaLocal(int linha, int coluna);
```

// Função: informar a posição do jogador no mapa (linha e coluna).

Ranking (utilizado em aulas práticas):

typedef struct {...}Ranking; // Função: informar o ranking do jogador.

2.3 Funções

void loadSounds(Sound som[NUM_SONS]) {

som[0] = LoadSound("assets/sounds/openChest.ogg");

} // Carrega os sons com a estrutura criada pela raylib. Podíamos ter implementado muito mais sons, porém não tivemos tempo. É uma função simples que usa a estrutura Sound da Raylib.

Localizacao criaLocal(int linha, int coluna) ; // Função que cria uma coordenada para o jogador.

void movVertical(Mapa *mapa, bool *animacaoVertical, int direcao, int *frameCounter, int *dMov); // Função responsável pela movimentação vertical do personagem que, caso o usuário escolha a movimentação de direita ou esquerda (int direcao), irá movimentar o jogador no mapa.

void movHorizontal(Mapa *mapa, bool *animacaoHorizontal, int direcao, int *frameCounter, int *dMov); // Função responsável pela movimentação horizontal do personagem que, caso o usuário escolha a movimentação de cima ou baixo (int direcao), irá movimentar o jogador no mapa.

void gravidade(Mapa *mapa, bool *caindo, int h, int *frameCounter, int *dMov, int *tMov, int *alt); // Função que permite a animação do personagem caindo, juntamente com fórmulas logicamente formadas para possibilitar essa ação.

void loadItens(Item item[NUM_ITENS]); // Função que cria itens, como por

exemplo os armazenados nos baús, joias para definir pontuação etc. Todos os seus respectivos itens estão com sua própria autodescrição.

void loadItens(Item item[NUM_ITENS]); // Função que carrega todos os itens.

Item criaItem(char nome[TAM_NOME_ITEM], char descricao[TAM_DESC], int tipo, int ID); // Função que cria os dados de um item.

Item getLevelItem(int fase, Item itens[NUM_ITENS]); // Função que faz com que o jogador receba o item especial da fase: na fase 1 é a espada, na fase 2 é o peitoral e a fase 3 é a maçã dourada.

Item getKey(Item itens[NUM_ITENS]); // Função que decide se o jogador adquiriu a chave para passar de fase. Se adquiriu, a porta para se teletransportar para a outra fase fica visível.

Item getRandomItem(int fase, Item item[NUM_ITENS]); // Função que consegue um item aleatório normal em cada baú (item que dá ponto ou poção de vida).

void jogCima(Mapa *mapa, bool *animacaoVertical, int *dMov, int *frameCounter);

// Função que movimenta o jogador para cima.

void jogBaixo(Mapa *mapa, bool *animacaoVertical, int *dMov, int *frameCounter); // Função que movimenta o jogador para baixo.

void jogEsquerda(Mapa *mapa, bool *animacaoHorizontal, int *dMov, int *frameCounter); // Função que movimenta o jogador para a esquerda no mapa.

void jogDireita(Mapa *mapa, bool *animacaoHorizontal, int *dMov, int *frameCounter); // Função que movimenta o jogador para a direita no mapa.

void abrePorta(Mapa *mapa); // Função que faz o jogador se teletransportar através das portas.

void abreBau(Mapa *mapa, Item item[NUM_ITENS]); // Função que abre o baú,

printa a informação do item adquirido e some com o baú aberto do mapa.

int altura(Mapa mapa); // Função que retorna a altura do jogador

void saveGame(Mapa mapa); // Função que salva o jogo no arquivo.

void loadGame(Mapa *mapa); // Função que carrega o jogo conforme o arquivo.

void loadTextures(Texturas *textura); // Função que carrega as texturas dos assets.

void desenhaBloco(Texture2D textura, int l, int c); // Função que desenha um bloco de uma textura que foi passada como parâmetro.

int jog_diminui_vida(Jogador *jogador); // Função que diminui a vida do jogador.

void jog_aumenta_pontuacao(Jogador *jogador, int pontos); // Função que aumenta a pontuação do jogador conforme o item que ele pegou.

void jog_le_nome(Jogador *jogador); // Função que lê o nome do jogador

void jog_print_info(Jogador jogador); // Função que veio da aula prática. Não foi implementada no jogo.

void loadGameMap(int fase, Mapa *mapa); // Função que carrega o mapa de cada fase.

Jogador jogInit(Mapa mapa, Texturas textura); // Função que inicializa o jogador e coloca todas as informações necessárias no novo jogo criado.

void desenhaMolduraMapa(); // Função que desenha a moldura vermelha envolta do mapa

void desenhaFundo(Texturas textura); // Função que desenha o fundo (background) do jogo.

void imprimeMapa(Mapa mapa, Texturas textura); // Função que imprime o mapa conforme as texturas.

void imprimeJog(Mapa mapa, Texturas textura); // Função que imprime o jogador no mapa.

void printJogInfo(Mapa mapa, Texturas textura); // Função que desenha todas as informações do jogador na tela, como skin, vidas etc.

void selecionaMenu(int selected, Mapa *mapa, Texturas textura); // Função que, dependendo do item selecionado no int selected (Novo Jogo, Carregar Jogo, etc.), chama a função certa corretamente.

void DesenhaMenu(int selected); // Função que desenha o menu inicial do jogo. O parâmetro selected decide qual ação o jogador escolheu: Novo Jogo, Carregar Jogo, Ranking ou Sair.

void DesenhaEncerramento(); // Função que desenha a tela final quando o jogador perde todas as vidas.

3. Bibliotecas extras

Não houve mais bibliotecas implementadas além da Raylib, junto com as aprendidas em aula.

4. Organização do trabalho na dupla

Criamos um repositório no GitHub para compartilhamento de arquivos, possibilitando assim o trabalho em equipe à distância. Em fins de semana, fizemos um trabalho em equipe presencial, onde foi decidido revezadamente qual integrante iria na casa do outro.

Em questão de programação, o integrante da dupla Thiago foi responsável pela realização dos seguintes feitos: integração do menu principal ao jogo, integração da tela final (quando o personagem perde todas as vidas), integração da música ao jogo, manipulação de assets, animação do personagem (que, por sinal, não pôde ser implementado no jogo por falta de tempo), compartilhamento de ideias criativas, compactar certas partes do código, manipulação de funções, implementação da tela de ranking do jogo e busca de funções da biblioteca Raylib para facilitar o trabalho do outro integrante.

Já o integrante Fernando foi responsável foi responsável por: integração de características do personagem ao jogo (como nome, pontos etc.), lógica de ponteiros, movimentação personalizada do personagem, manipulação de funções, lógica matemática para funções complexas (como a gravidade), criação da maioria das funções e pelo trabalho em funções repetitivas (como a implementação de todos os assets).

Os dois integrantes fizeram muito mais do que o descrito. Em destaque, o Fernando houve facilidade na manipulação do código, enquanto o Thiago, pela criatividade e facilidade na resolução de problemas.

5. Desafios enfrentados e aprendizados adquiridos

Ao longo da arquitetura do projeto, foi aprendido a utilização de uma nova biblioteca, possibilitando retomar e fixar o conteúdo passado ao longo do semestre.

O maior desafio enfrentado foi conciliar o tempo e completar o projeto no prazo certo. Em questão de programação, não obtivemos dificuldade ao longo do semestre.

6. Referências

Tiramos todas as bibliotecas usadas no programa do site da Raylib disponível para consulta: https://www.raylib.com/cheatsheet/cheatsheet.html

O roteiro do jogo se passa equivalentemente ao jogo Minecraft.