UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JULIA MORO NUNES
KARINA RIBEIRO MARTINS
SAYID RAYHAN ABDEL GADER SAFA

RELATÓRIO FINAL DO TRABALHO DE ALGORITMOS E PROGRAMAÇÃO
JOGO BANANA DEFENSE

Porto Alegre, 22 de agosto de 2024.

Descrição:

O jogo Banana Defense consiste em uma aplicação inspirado nas mecânicas de jogos do gênero “Tower Defense”. Ele foi desenvolvido em linguagem C e utiliza a biblioteca Raylib, conforme os requisitos. O projeto pode ser visualizado no Github através do link: https://github.com/Sayidyuyu/bananaDefense

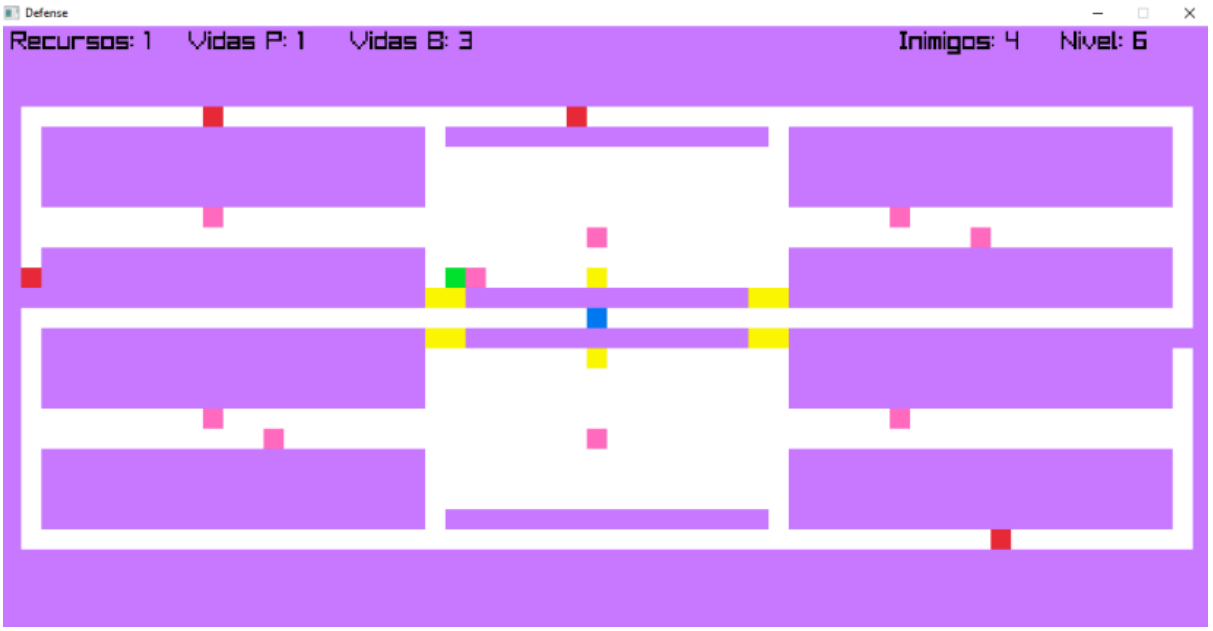
Os arquivos de código do jogo estão estruturados da seguinte forma:

Pasta ‘include’:
estruturas.h -> contém todas as *structs,* constantes e inclusão de bibliotecas.
funcoes.h -> contém as funções auxiliares desenvolvidas por nós.

Pasta ‘src’:
main.h -> contém exclusivamente a função main.

Nas telas estáticas (menu, pause, vitória, *gameover*) é indicado para o usuário as opções que ele pode selecionar, bem como as teclas que correspondem a cada uma.
Segue uma tela de exemplo:

Durante o jogo ativo, a tela parecida com a abaixo, de acordo com a fase atual:



Os quadrados são divididos da seguinte forma:

Branco	->	Área transitável
Roxo	->	Parede
Verde	->	Jogador
Vermelho	->	Inimigo
Amarelo	->	Buraco
Rosa	->	Recurso
Preto	->	Obstáculo

Os elementos interativos se comportam da seguinte maneira:

1. O jogador pode se mover através das setas do teclado, ou das teclas WASD.
2. Os inimigos se movem sempre em direção à base.
3. Nenhuma das entidades pode atravessar as paredes.
4. Apenas o jogador pode atravessar os buracos e coletar os recursos.
5. Cada recurso coletado dá ao jogador o direito de colocar um obstáculo no mapa, que pode ser feito pela tecla G.
6. Ao colidir com um obstáculo, o inimigo morre.
7. Se o jogador colidir com um inimigo, ele perde uma vida.
8. Se um inimigo colidir com a base, ela perde uma vida.
9. Se o jogador ou a base perderem todas as vidas, o jogo acaba, e o jogador falha.
10. Se o jogador matar todos os inimigos, o nível é concluído com sucesso.
11. O jogo pode ser pausado ao pressionar a tecla TAB.
12. O jogo pode ser abandonado ao pressionar a tecla ESC.

No código, as estruturas são definidas de forma modular. Há uma estrutura COORDENADAS, que armazena as coordenadas (x, y) de uma entidade, bem como seu deslocamento (dx, dy).

Cada entidade (jogador, inimigos e base) conta com uma estrutura própria. Ela contém um elemento do tipo COORDENADAS, além de outras variáveis específicas de cada entidade.

As funções utilizadas são as seguintes:

```
//--------------------------------------------------------------------------------
// Protótipos das funções
//--------------------------------------------------------------------------------

// Inicializa o player com valores base, como vida, cor, letra, recursos, delay e timer
void inicializaPlayer(TIPO_PLAYER *player);
// Inicializa o inimigo com valores base, como vida, cor, letra, timer e coordenadas
void inicializaInimigo(TIPO_INIMIGO *inimigo);
// Inicializa a base com valores base, como vida, cor e coordenadas
void inicializaBase(BASE *base);
// Verifica a quantidade de vidas da base e do player
void verificaVidas(BASE *base, TIPO_PLAYER *player, GAMESCREEN *tela);
// Desenha o player na tela com posicao x e y
void desenhaPlayer(TIPO_PLAYER *player, int posx, int posy);
// Desenha o inimigo na tela com posicao x e y
void desenhaInimigo(TIPO_INIMIGO inimigo[MAX_INIMIGOS], int dx, int dy);
// Conta a quantidade de recursos coletados pelo player
void contaRecursos(TIPO_PLAYER *player);
// Verifica se o player deve mover, retorna 1 se deve mover, 0 se nao deve mover
int deveMoverPlayer(TIPO_PLAYER *entidade, char *matriz);
// Verifica se o inimigo deve mover, retorna 1 se deve mover, 0 se nao deve mover
int deveMoverInimigo(TIPO_INIMIGO *inimigo, TIPO_PLAYER *player, char *matriz, BASE *base, int *qtdInimigo);
```

```
// Move a entidade no mapa com a letra correspondente e coloca espaco em branco na posicao anterior
void move(COORDENADAS *entidade, char *matriz, char letra);
// Controle do jogador, verifica teclas pressionadas e movimenta o jogador
void controleJogador(TIPO_PLAYER *entidade, char *matriz);
// Move o inimigo no mapa
void moveInimigo(TIPO_INIMIGO *inimigo, TIPO_PLAYER *player, char *matriz, BASE *base, int *qtdInimigo);
// Redefine o deslocamento do inimigo
void redefineDeslocamentoInimigo(TIPO_INIMIGO *inimigo, TIPO_PLAYER *player, char *matriz, BASE *base, int *qtdInimigo);
// Centraliza a janela do jogo ao centro da tela
void centerWindow(float windowWidth, float windowHeight);
// Controle do jogador, verifica teclas pressionadas e movimenta o jogador
void controleJogador(TIPO_PLAYER *entidade, char *matriz);
// Verifica a quantidade de vidas da base e do player e muda a tela de acordo
void verificaVidas(BASE *base, TIPO_PLAYER *player, GAMESCREEN *tela);
// Verifica a tela atual do jogo e muda conforme a tecla pressionada e a quantidade de inimigos
void verificaTelaJogo(char *matriz, GAMESCREEN *telaAtual, int *deveFechar, GAMESTATUS *estadoDoJogo, char fase[], TIPO_INIMIGO inimigos[], TIPO_PLAYER *player, BASE *base, int *qtdInimigos);
// Desenha o mapa do jogo com o player, inimigos e base
void desenhaMapa(char *matriz, TIPO_PLAYER *player, TIPO_INIMIGO inimigo[MAX_INIMIGOS], BASE *base);
// Inicializa o nivel do jogo com valores padrao e leitura do arquivo de fases
void inicializaNivel(char *matriz, char *fase, GAMESTATUS *estadoDoJogo, TIPO_INIMIGO inimigos[], TIPO_PLAYER *player, BASE *base, int *qtdInimigos);
```