Universidade Federal de Minas Gerais

Documentação r-type - Allegro

Aline Cristina Pinto
Professor: Pedro Olmo Stancioli Vaz de Melo

Belo Horizonte
2021/2

1 - Introdução

1.1 Descrição do jogo

Este trabalho tem como objetivo desenvolver uma versão básica do jogo arcade R-Type. Na presente versão, o jogador controla a nave *R-9A Arrowhead* no espaço enquanto desvia de *blocos* e *bydo minions*. Diferentemente dos blocos, os inimigos *Gouache, Fenrir* e *Ollie* podem ser destruídos com o auxílio de dois tipos mísseis: o tiro básico (que é destruído na colisão) e o tiro avançado (capaz de destruir vários inimigos sequencialmente desde que estejam na sua área de alcance). Ainda, o jogador é pontuado por cada inimigo destruído: o tipo do inimigo e a velocidade dos mesmo são os parâmetros utilizados para o cálculo desse *score*.

1.2 Como jogar

Movimentação da nave:

W: Utilizado para mover a nave para cima.
S: Utilizado para mover a nave para baixo.
D: Utilizado para mover a nave para a direita.
A: Utilizado para mover a nave para a esquerda.

Controle de tiro:

Básico: Pressionar a tecla Espaço rapidamente.
Avançado: Pressionar a tecla Espaço até que o raio do tiro se estabilize em seu valor máximo.

2 - Implementação

2.1 /interfaces

scenario_interface:
Contém as entidades *Star* para as estrelas que compõem o espaço e *Score* que armazena os dados de pontuação e estatística do jogo. Além disso, possui as constantes relacionadas a quantidade de estrelas.

spaceship_interface:
Contém as entidades *Spaceship* para nave e *Projectile* para os tiros. Além disso, possui as constantes e enums relacionadas a velocidade, posições e dimensões dos mesmos.

obstacles_interface:
Contém as entidades *Block* para os blocos e *Enemy* para os inimigos. Além disso, possui as constantes e enums relacionadas a velocidade, posições e dimensões dos bydo minions e do bloco.

2.2 /ui

A seção de ui foi subdividida por domínios que conversam diretamente com as interfaces definidas anteriormente. Ainda, possuem as funções de desenho, atualização e controle de colisões. As colisões foram feitas utilizando o conceito de ***Bounding box collision*** que considera todos os elementos como quadriláteros (quadrado ou retângulo).

scenario

scenario.h/scenario.c

- *initStars(Star stars[], float moveSpeed)*: Inicia o vetor de estrelas e permite a mudança de velocidade. Esse segundo parâmetro foi disponibilizado para a geração de diferentes planos de estrelas com velocidades diferentes.
- *updateStars(Star stars[])*: Atualiza o vetor de estrelas decrementado de x para a movimentação da direita para a esquerda.
- *drawStars(Star stars[])*: Desenha as estrelas na tela.

- *initScore(Score *score)*: Inicia o score zerando todos os atributos,o preenchimento do recorde anterior é feito posteriormente.
- *updateScore(Score *score)*: Atualiza o recorde e a flag de novo recorde caso a pontuação atual seja maior do que a obtida no início do jogo.
- *drawScore(Score score, ALLEGRO_FONT *font22)*: Desenha a pontuação do jogo no canto superior esquerdo.
- *drawGameOver(Score score, ALLEGRO_FONT *font22, ALLEGRO_FONT *font26)*: Desenha a mensagem final com a pontuação obtida, o recorde e as estatísticas do jogo (quantidade de inimigos mortos).

spaceship

spaceship.h/spaceship.c

- *initSpaceship(Spaceship *spaceship)*: Inicia a nave com seus atributos default.
- *drawSpaceship(Spaceship spaceship)*: Desenha a nave utilizando regions para construir o *spirit animation*.
- *updateSpaceship(Spaceship *spaceship)*: Atualiza x e y para a movimentação da nave. Nessa função há a validação que impede que a nave saia dos limites da tela.

- *controlSpaceship(int keycode, Spaceship *spaceship, Projectile *projectile, KeyEventsEnum keyEvent)*: Possui o switch case com a identificação dos eventos de clique que permitem a movimentação da nave e a animação da nave. Além disso, controla os tipos de tiro através de flags no evento da tecla espaço.

- *initProjectile(Projectile *projectile, Spaceship spaceship)*: Inicia o tiro básico um pouco à frente da nave.
- *resetProjectile(Projectile *projectile, Spaceship spaceship)*: Chama a função de init para resetar o tiro.

- *updateProjectile(Projectile *projectile, Spaceship spaceship)*: Atualiza o tiro realizando verificações para tiro fora da tela, tiro avançado e incremento para movimentação.
- *drawProjectile(Projectile projectile)*: Desenha tiro na tela.
- *hasCollisionBetweenProjectileAndEnemies(Projectile *projectile, Enemy enemy)*: Verifica se houve colisão entre tiro e inimigos retornando 1 caso verdadeiro e 0 caso falso. (Método privado)
- *handleScore(Score *score, Enemy enemy)*: Realiza o cálculo da pontuação de acordo com o inimigo e sua velocidade e atualiza o objeto de score. (Método privado)
- *handleCollisionBetweenProjectileAndEnemies(Projectile *projectile, Spaceship spaceship, Score *score, Enemy enemies[])*: Lida com a colisão entre tiros e inimigos utilizando os métodos privados anteriores para isso.
- *handleCollisionBetweenProjetileAndBlock(Projectile *projectile, Spaceship spaceship, Block block)*: Lida com a colisão entre tiros e blocos, atualizando o tiro como não ativo.

obstacles

block.h/block.c e *enemies.h/enemies.c*

block.h/block.c

- *initBlock(Block *block)*: Cria bloco com dimensões e velocidade aleatórias.
- *updateBlock(Block *block)*: Atualiza x do bloco para movimentação. Além disso, checa se o bloco não está na tela para reiniciá-lo.
- *drawBlock(Block block)*: Desenha bloco na tela (imagem).
- *spaceshipAndBlockCollision(Spaceship spaceship, Block block)*: Verifica se há colisão entre nave e bloco. Esse método retorna um booleano que será utilizado para finalizar o jogo.

enemies.h/enemies.c

- *sortEnemyType()*: Retorna um tipo de inimigo aleatório para a criação do vetor de inimigos. (Método privado)
- *getEnemyTypePosition(EnemyTypeEnum type)*: Retorna as posições corretas dado o tipo de inimigo para criação do *spirit*. (Método privado)
- *getEnemyTypeWidth(EnemyTypeEnum type)*: Retorna comprimento correto dado o tipo de inimigo para criação do *spirit*. (Método privado)
- *getEnemyTypeHeight(EnemyTypeEnum type)*: Retorna altura correta dado o tipo de inimigo para criação do *spirit*. (Método privado)
- *initEnemies(Enemy enemies[], ALLEGRO_BITMAP *image)*: Inicia vetor de inimigos utilizando os métodos privados anteriores para a aleatoriedade dos mesmos.
- *releaseEnemies(Enemy enemies[])*: Ativa inimigos na tela.
- *updateEnemies(Enemy enemies[])*: Atualiza x dos inimigos para a movimentação e desativa inimigos que não estão na tela para liberá-los novamente.
- *drawEnemies(Enemy enemies[])*: Desenha inimigos (imagem).

- *hasCollisionBetweenEnemies(Enemy enemy1, Enemy enemy2)*: Verifica se houve colisão entre inimigos retornando 1 caso verdadeiro e 0 caso falso. (Método privado)
- *handleCollisionBetweenEnemies(Enemy enemies[])*: Atualiza inimigos que colidem entre si como não ativos utilizando o método privado anterior.
- *hasCollisionBetweenEnemiesAndBlock(Enemy enemy1, Block block)*: Verifica se houve colisão entre inimigos e bloco retornando 1 caso verdadeiro e 0 caso falso. (Método privado)
- *handleCollisionBetweenEnemiesAndBlock(Enemy enemies[], Block block)*: Atualiza inimigos que colidem com o bloco como não ativos utilizando o método privado anterior.
- *spaceshipAndEnemiesCollision(Spaceship spaceship, Enemy enemies[])*: Verifica se há colisão entre nave e inimigos. Esse método retorna um booleano que será utilizado para finalizar o jogo.

2.3 /styles

colors.h/colors.c:
Contém a abstração de algumas cores que podem ser utilizadas no jogo.

2.4 /utils

constants.h:
Contém as constantes gerais do jogo: SCREEN_H, SCREEN_W e FPS

utils.h/utils.c:

- *openFile(char* name, char* mode)*: Abstração para abertura de arquivo.
- *closeFile(FILE* file)*: Abstração para fechar arquivo.
- *readGameHistory(FILE* file, Score *score)*: Atualiza os dados de pontuação dado o arquivo de recorde.
- *writeGameHistory(FILE* file, Score score)*: Sobrescreve arquivo dado informações de pontuação.

2.5 Arquivo principal

rtype.c:

- *getGameHistory(Score *score)*: Controla utilitários de abertura e leitura de arquivo para preencher dados de pontuação no início do jogo.
- *updateGameHistory(Score score)*: Escreve em arquivo caso a pontuação seja maior que o recorde anterior.
- *endGame(Score score, ALLEGRO_FONT *font22, ALLEGRO_FONT *font26)*: Gerencia a atualização de arquivo e criação de tela de *game over*.

- *int hasBoundingBoxCollision(float x1, float y1, float width1, float height1, float x2, float y2, float width2, float height2)*: Verifica a colisão entre dois elementos dada suas coordenadas e dimensoes considerando os mesmos como boxes.

- *main(int argc, char **argv)*: Instância os addons e controle do jogo, chamando as funções anteriores no while principal.

2.6 /assets

img:
As imagens foram obtidas no site (https://www.spriters-resource.com/snes/superrtype/) e movificadas no paint.

fonts:
AtariSmall.ttf: https://fontzone.net/font-details/atari-small

2.7 Código

https://github.com/alinecristinapinto/rtype