Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

Kosmostars

ОТЧЁТ

по командному заданию

Авторы работы:

студенты группы 21312

Савин И. А.

Шильников Е.И.

Кострыкин В. А.

Петрозаводск 2018

# Цель работы

Нужно на С++ (в Visual Studio 2010 Express) с использованием SFML (Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека) написать простую 2D игру. При разработке программы должно быть использовано наследование, контейнеры, итераторы, раздельная компиляция. Промежуточные результаты работы команды должны сохраняться на GitHub.

# Описание сюжета игры

Пользователь с помощью стрелочек вверх и вниз управляет кораблём, летящим в космосе. Навстречу кораблю летят астероиды, против которых можно использовать лазер (кнопка пробел), чтобы их разрушить, так как если астероид вылетит за границу карты или врежется в корабль, то пользователь теряет очки. Есть три вида астероидов: средние и большие, которые появляются сами по себе с вероятностью 0.7 и 0.3 соответственно и маленькие, которые появляются при взрыве большого и разлетаются в разные стороны со случайным углом, что сильно усложняет игру, но у этих астероидов есть и плюс!   
 В одном из пяти случаев из маленького астероида может выпасть один из видов бонусов: уничтожитель, 2х или щит. Уничтожитель взорвёт все астероиды, имеющиеся на экране, 2х на короткое время повышает количество полученных очков в 2 раза, но с этим бонусом стоит быть очень внимательным, так как и снимаемые очки увеличиваются вдвое! И последний бонус – щит, он даёт пользователю полную защиту от траты очков на короткое время.  
 Задача пользователя - набрать как можно больше очков.

# Программная реализация

Проект разрабатывался в среде MS Visual Studio 2010 Express на языке C++

Единицы компиляции:

1. Entity.h и Entity.cpp – базовый класс для всех геёмплейных объектов. Содержит в себе перечисление статусов и «баффов»
2. Player.h / Player.cpp – класс корабля игрока. Содержит методы для перемещения за указателем мыши и стрельбы.
3. Missile.h / Missile.cpp – класс ракеты. Содержит данные о наносимом при попадании уроне и размере взрыва. Логика движения и взрыва обрабатывается базовым классом
4. Enemy.h / Enemy.cpp – класс вражеского корабля. Содержит логику появления с задержкой, периодической стрельбы и появления чайки. При приближении к краю экрана меняет направление движения в горизонтальной оси.
5. Bird.h / Bird.cpp – класс чайки. Содержит логику анимации и зеркального отражения от краев экрана. Количество жизней чайки уменьшается со временем, после чего она взрывается.
6. main.cpp – точка входа С++ int main()
7. Game.h / Game.cpp – основной класс игры. Содержит:
   * RenderWindow\* window – окно игры
   * Texture background – текстура фона
   * Player player – объект игрока
   * list<Enemy\*> enemies – список врагов
   * const int ENEMY\_COUNT – максимальное количество врагов
   * int enemiesCount – текущее количество врагов
   * конструктор Game() для инициализации вышеназванных объектов
   * метод start():
8. добавление врагов на карту
9. обработка событий окна
10. циклические отрисовка, обновление, движение и поиск столкновений объектов игры
11. удаление взорванных / улетевших объектов из динамической памяти

# Заключение

Проект был разработан в среде MS Visual Studio 2010 Express с использованием SFML и системы контроля версий Git. Все прецеденты реализованы. Сбои и зависания не наблюдается. Использована раздельная компиляция. Реализована очистка динамической памяти, неиспользуемые переменные отсутствуют. Приложены диаграммы вариантов использования и классов, исходный код программы.

# Приложение 1

Диаграмма вариантов использования

Диаграмма классов

# Приложение 2

Asteroid.h

#pragma once

#include "Entity.h"

class Asteroid:public Entity{

private:

int reward;

Texture explosionTexture;

Sprite explosionSprite;

float explosionFrame;

float scale;

Vector2f speed;

Status status;

public:

int hp;

Asteroid(Image & image, Image & explosionImage);

Asteroid(Image & image, Image & explosionImage, float x, float y, float speedX, float speedY);

Status update(float time);

void setScale();

float getScale();

void getDamage();

int getReward();

Sprite explosion(float time);

};

Asteroid.cpp

#include "Asteroid.h"

Asteroid::Asteroid(Image& image, Image& explosionImage)

{

explosionFrame = 0;

setScale();

hp = 4 \* scale;

speed.x = 2 / scale;

speed.y = 0;

reward = 12 \* scale;

makeTexture(image, rand() % 3, 0, scale);

setPosition(900, rand() % (540 - size.x));

explosionTexture.loadFromImage(explosionImage);

explosionSprite.setTexture(explosionTexture);

explosionSprite.setScale(scale, scale);

}

Asteroid::Asteroid(Image& image, Image& explosionImage, float x, float y, float speedX, float speedY)

{

explosionFrame = 0;

scale = 0.25;

hp = 1;

speed.x = speedX;

speed.y = speedY;

reward = 3;

makeTexture(image, rand() % 3, 0, scale);

setPosition(x, y);

explosionTexture.loadFromImage(explosionImage);

explosionSprite.setTexture(explosionTexture);

explosionSprite.setScale(0.75, 0.75);

}

Status Asteroid::update(float time)

{

if (status == dead)

return status = exploding;

if (status == alive) {

position.x -= speed.x \* time \* 500;

position.y -= speed.y \* time \* 500;

sprite.setPosition(position);

}

if (explosionFrame >= 10)

return status = exploded;

if (status == exploding)

return status = exploding;

if (position.x < -96 || position.x > 960 || position.y > 540 || position.y < -96)

return status = outboard;

if (hp <= 0)

return status = dead;

return status = alive;

}

void Asteroid::setScale()

{

scale = (rand() % 100 < 70) ? 0.5 : 1;

}

void Asteroid::getDamage()

{

hp--;

}

float Asteroid::getScale()

{

return scale;

}

int Asteroid::getReward()

{

return reward;

}

Sprite Asteroid::explosion(float time)

{

int x = (int)explosionFrame % 5;

int y = (int)(explosionFrame / 5);

explosionSprite.setTextureRect(IntRect(x \* 96, y \* 96, 96, 96));

explosionSprite.setPosition(position.x, position.y);

explosionFrame += time \* 100;

return explosionSprite;

}

Bonus.h

#pragma once

#include "Entity.h"

class Bonus : public Entity {

public:

Buff buff;

Bonus(Image& image, Vector2f playerPose);

Status update(float time);

bool isAlive();

Buff getBuff();

};

Bonus.cpp

#include "Bonus.h"

Bonus::Bonus(Image& image, Vector2f playerPose)

{

makeTexture(image, buff = static\_cast<Buff>(rand() % 3), 0, 0.5);

setPosition(playerPose.x + 16, playerPose.y + 16);

}

Status Bonus::update(float time)

{

position.x -= time \* 1000;

sprite.setPosition(position);

if (position.x < -96) {

return outboard;

}

return stay;

}

Buff Bonus::getBuff()

{

return buff;

}

Entity.h

#pragma once

#include <SFML/Graphics.hpp>

enum Buff { demolisher,

x2,

shield,

nobuff };

enum Status { del,

alive,

exploding,

exploded,

outboard,

shoot,

stay,

dead,

DEMOLISH };

using namespace sf;

class Entity {

protected:

Texture texture;

Sprite sprite;

Status status;

public:

Vector2f position;

Vector2u size;

void makeTexture(Image& image, int x, int y, float scale = 1);

void makeTexture(Image& image);

void setPosition(float x, float y);

Sprite getSprite();

bool intersects(Entity\* obj);

Status update(float);

};

Entity.cpp

#include "Entity.h"

void Entity::makeTexture(Image& image, int x, int y, float scale)

{

texture.loadFromImage(image); // загружаем изображение игрока

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(x \* 96, y \* 96, 96, 96));

sprite.setScale(scale, scale);

size = texture.getSize();

size.x \*= scale;

size.y \*= scale;

}

void Entity::makeTexture(Image& image)

{

texture.loadFromImage(image); // загружаем изображение игрока

sprite.setTexture(texture);

size = texture.getSize();

}

void Entity::setPosition(float x, float y)

{

position.x = x;

position.y = y;

sprite.setPosition(position);

}

Sprite Entity::getSprite()

{

return sprite;

}

bool Entity::intersects(Entity\* obj)

{

return this->getSprite().getGlobalBounds().intersects(

obj->getSprite().getGlobalBounds());

}

Game.h

#pragma once

#include <string>

#include <list>

#include <iostream>

#include "Entity.h"

#include "Map.h"

#include "Player.h"

#include "Laser.h"

#include "Asteroid.h"

#include "Bonus.h"

using namespace sf;

class Game {

VideoMode desktop;

RenderWindow window;

Font font;

Text text;

Clock clock;

Player\* player;

Map\* map;

Image asteroidImage;

Image laserImage;

Image bonusImage;

Image explosionImage;

Image playerImage;

Image mapImage;

std::list<Asteroid\*> asteroids;

std::list<Laser\*> lasers;

std::list<Bonus\*> bonuses;

public:

Game();

~Game();

void play();

};

Game.cpp

#include "Game.h"

#include <cstdlib>

#include <ctime>

Game::Game()

{

srand(time(NULL));

desktop = VideoMode::getDesktopMode();

window.create(VideoMode(960, 540, desktop.bitsPerPixel), "COSMOSTARS");

font.loadFromFile("FONT.ttf"); //передаем нашему шрифту файл шрифта

text.setCharacterSize(40);

text.setColor(Color::Red); //покрасили текст в красный

text.setFont(font);

asteroidImage.loadFromFile("images/asteroids.png");

laserImage.loadFromFile("images/bullet.png");

bonusImage.loadFromFile("images/bonuses.png");

explosionImage.loadFromFile("images/explosion.png");

playerImage.loadFromFile("images/ship.png");

mapImage.loadFromFile("images/cosmos.png");

player = new Player(playerImage);

map = new Map(mapImage);

}

Game::~Game() {}

void Game::play()

{

float asteroid\_cooldown = 0;

while (window.isOpen() && player->getStatus() != dead) {

Event event;

while (window.pollEvent(event))

if (event.type == Event::Closed)

window.close();

window.clear();

float time = clock.getElapsedTime().asSeconds();

window.draw(map->getSprite());

window.draw(player->getSprite());

map->update(time);

switch (player->update(time)) {

case shoot:

lasers.push\_back(new Laser(laserImage, player));

break;

case DEMOLISH:

for (auto asteroid = asteroids.begin(); asteroid != asteroids.end(); asteroid++)

(\*asteroid)->hp = 0;

break;

}

for (auto bonus = bonuses.begin(); bonus != bonuses.end();) {

window.draw((\*bonus)->getSprite());

if ((\*bonus)->update(time) == outboard) {

delete \*bonus;

bonus = bonuses.erase(bonus);

}

else if (player->intersects(\*bonus)) {

player->setBuff((\*bonus)->getBuff());

delete \*bonus;

bonus = bonuses.erase(bonus);

}

else

bonus++;

}

for (auto laser = lasers.begin(); laser != lasers.end();) {

window.draw((\*laser)->getSprite());

if ((\*laser)->update(time) == del) {

delete \*laser;

laser = lasers.erase(laser);

}

else {

for (auto asteroid = asteroids.begin(); asteroid != asteroids.end(); asteroid++)

if ((\*laser)->intersects(\*asteroid)) {

(\*asteroid)->getDamage();

(\*laser)->hit();

}

laser++;

}

}

asteroid\_cooldown += time;

if (asteroid\_cooldown > 0.25) {

asteroids.push\_back(new Asteroid(asteroidImage, explosionImage));

asteroid\_cooldown = 0;

}

for (auto asteroid = asteroids.begin(); asteroid != asteroids.end();) {

switch ((\*asteroid)->update(time)) {

case outboard:

player->setScores(-(\*asteroid)->getReward());

delete \*asteroid;

asteroid = asteroids.erase(asteroid);

continue;

case exploding:

window.draw((\*asteroid)->explosion(time));

asteroid++;

continue;

case exploded:

if ((\*asteroid)->hp != 0)

player->setScores((\*asteroid)->getReward());

delete \*asteroid;

asteroid = asteroids.erase(asteroid);

continue;

case dead:

if ((\*asteroid)->getScale() == 1) {

int x = rand() % 4 + 1;

int y = rand() % 4 + 1;

asteroids.push\_front(new Asteroid(asteroidImage, explosionImage, (\*asteroid)->position.x + 24, (\*asteroid)->position.y + 24, x, y));

asteroids.push\_front(new Asteroid(asteroidImage, explosionImage, (\*asteroid)->position.x + 24, (\*asteroid)->position.y + 48, x, -y));

asteroids.push\_front(new Asteroid(asteroidImage, explosionImage, (\*asteroid)->position.x + 48, (\*asteroid)->position.y + 24, -x, y));

asteroids.push\_front(new Asteroid(asteroidImage, explosionImage, (\*asteroid)->position.x + 48, (\*asteroid)->position.y + 48, -x, -y));

}

if ((\*asteroid)->getScale() == 0.25 && rand() % 5)

bonuses.push\_back(new Bonus(bonusImage, (\*asteroid)->position));

asteroid++;

continue;

case alive:

if (player->intersects(\*asteroid)) {

player->setScores(-2 \* (\*asteroid)->getReward());

(\*asteroid)->hp = 0;

}

else

window.draw((\*asteroid)->getSprite());

asteroid++;

continue;

}

}

text.setString("SCORES: " + player->getScores());

text.setPosition(000, 0);

window.draw(text);

if (player->getBuff() != nobuff) {

std::string buff;

if (player->getBuff() == x2)

buff = "X2";

if (player->getBuff() == shield)

buff = "SHIELD";

text.setString(buff + " TIME: " + player->getBuffTimeLeft());

text.setPosition(500, 0);

window.draw(text);

}

window.display();

clock.restart();

}

if (player->getStatus() == dead) {

int i = 0;

while (i++ < 1000)

std::cout << "YOU LOSE ";

}

}

Laser.h  
#pragma once

#include "Player.h"

#include "Entity.h"

class Laser : public Entity {

public:

Laser(Image& image, const Player\* player);

Status update(float time);

void hit();

};

Laser.cpp

#include "Laser.h"

Laser::Laser(Image& image, const Player\* player)

{

makeTexture(image);

setPosition(player->position.x + player->size.x,

player->position.y + player->size.y / 2 - this->size.y / 2);

}

Status Laser::update(float time)

{

if (status != del) {

position.x += 50 \* time \* 500;

sprite.setPosition(position);

status = (position.x > 960) ? del : stay;

}

return status;

}

void Laser::hit() { status = del; }

Map.h

#pragma once

#include "Entity.h"

class Map : public Entity {

private:

float displace;

public:

Map(Image& image);

Status update(float time);

bool isAlive();

};

Map.cpp

#include "Map.h"

Map::Map(Image& image)

{

makeTexture(image);

displace = 0;

}

Status Map::update(float time)

{

displace += time \* 1000;

if (displace >= 960)

displace = 0;

setPosition(-displace, 0);

return stay;

}

Player.h

#pragma once

#include "Entity.h"

class Player : public Entity {

private:

int scores;

float shot\_cooldown;

float buffTime;

Buff buff;

public:

Player(Image& image);

Status update(float time);

void setBuff(Buff buff);

std::string getScores();

void setScores(int crement);

Status getStatus();

Buff getBuff();

std::string getBuffTimeLeft();

};

Player.cpp

#include "Player.h"

#include <sstream>

Player::Player(Image& image)

{

scores = 100;

shot\_cooldown = 0;

makeTexture(image);

setPosition(0, 270);

buffTime = 0;

buff = nobuff;

}

Status Player::update(float time)

{

if (scores <= 0)

return status = dead;

shot\_cooldown += time;

if (getBuff() == demolisher) {

setBuff(nobuff);

return DEMOLISH;

}

if (getBuff() != nobuff)

buffTime += time;

if (buffTime >= 1) {

setBuff(nobuff);

buffTime = 0;

}

if (position.y > 0 && Keyboard::isKeyPressed(Keyboard::Up))

position.y -= 0.5 \* time \* 10000;

else if (position.y + size.y < 540 && Keyboard::isKeyPressed(Keyboard::Down))

position.y += 0.5 \* time \* 10000;

else if (Keyboard::isKeyPressed(Keyboard::Space) && shot\_cooldown > 0.01) {

shot\_cooldown = 0;

return status = shoot;

}

sprite.setPosition(position);

return status = alive;

}

void Player::setBuff(Buff buff)

{

this->buff = buff;

buffTime = 0;

}

std::string toStr(float f)

{

std::stringstream ss;

ss << f;

return ss.str();

}

std::string Player::getScores()

{

return toStr(scores);

}

std::string Player::getBuffTimeLeft()

{

return toStr(1 - buffTime);

}

void Player::setScores(int crement)

{

switch (buff) {

case x2:

scores += 2 \* crement;

break;

case shield:

if (crement > 0)

scores += crement;

break;

default:

scores += crement;

}

}

Status Player::getStatus()

{

return status;

}

Buff Player::getBuff()

{

return buff;

}

Main.cpp

#include "Game.h"

int main(){

Game game;

game.play();

}