**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Физико-технический институт**

**Командное задание в рамках курсового проекта «Разработка игры «Space invaiders»**

**Выполнили:**

Студенты

Физико-технического

Института группы 21318

Иванов Константин

ПереверзеваДарья

Сергиенко Данил

Пиялкин Денис

Шимозёров Илья

**Преподаватель:**

Бульба Артем Владимирович

**Петрозаводск 2023**

**Цель работы:** Разработать аналог игры «space invaiders». Вести разработку с помощью системы контроля версий Git через веб-сервис GitHub. Закрепить полученные знания, использования языков С++ с использованием библиотеки

SFML и UML.

**Задачи:**

1. Познакомится с принципами работы в команде.
2. Научится использовать при работе систему контроля версии Git.
3. Научится строить UML-диаграммы.
4. Научится использовать SFML библиотеки в программной среде С++.
5. Написание отчёта о проделанной работе.

Кратко о программной реализации:

Программная реализация выполнена на языке c++ с использованием библиотеки SFML, среда разработки - IDE Qt Creator (Qt 5.4.2)

Единицы компиляции:

Globals.h

создан для хранения констант и более удобного управления их значениями в целях отладки

Entity.h

Entity.cpp

Содержат абстрактный класс, описывающий общее поведение объектов в рамках игры

Bullet.h

Bullet.cpp

Создание пули, метод взаимодействия с ними, а также их изменение со временем

Enemy.h

Enemy.cpp

Описание класса враг и его проведение как отдельного объекта

Player.h

Player.cpp

Описание класса игрока. В нем содержатся метод управления самолетом, список пуль, выпущенных игроком, и проверку на попадание по самолету

Enemies.h

Enemies.cpp

Описание класса враги, то есть список врагов и их поведение как одно целое. Отвечает за передвижение списка врагов по карте, обработке их смерти при попадании пули игрока по ним

Game.h

Game.cpp

Описание класса игры, собирающий все выше перечисленные классы вместе и обеспечивающий взаимодействие между ними

Описание сюжета игры

Игрок управляет самолетом, передвигая его горизонтально, в нижней части экрана, а также отстреливая инопланетян, надвигающихся сверху экрана. Целью игры является уничтожение всех инопланетян на экране, которые двигаются горизонтально, а также вертикально, по направлению к низу экрана. Игрок имеет бесконечное количество патронов. Попадая в инопланетянина, игрок уничтожает его, за что получает очки. При уничтожении инопланетян увеличивается скорость движения оставшихся. При уничтожении всех инопланетян появляется новая волна. Количество новых волн инопланетян неограниченно, что делает игру бесконечной.

Инопланетяне пытаются уничтожить самолет, стреляя по нему. При попадании в самолет он уничтожается, то есть игра заканчивается. При достижении хотя бы одним из противников нижней части экрана происходит инопланетный захват, и игра также заканчивается.

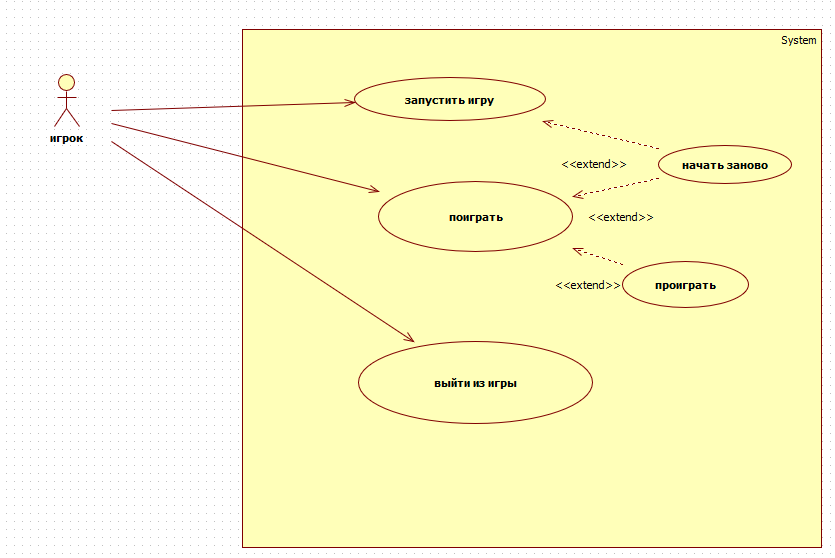


Рис1. Диаграмма вариантов использования

Запустить игру

Игрок запускает игру

Поиграть

С помощью стрелок игрок передвигает самолетик, уворачиваясь от пуль противника, а также стреляет в ответ.

Выйти из игры

Игрок выходит из игры

Проиграть

Если игрок не сможет увернутся от вражеской пули, игроку покажут его счет

Начать заново

При желании, игрок может начать игру заново сбросив свой счет и восстановив убитых врагов

Список существительных:

* Игра

Основной класс программы, в нем хранятся экземпляры классов игрока и врагов, а также обрабатывается основная логика игры, например проигрыш игрока

* Сущность

Абстрактный класс, представляющий собой объект на экране

* Игрок

Класс, представляющий игрока. Он отвечает за передвижение, выстрелы самолета и отслеживание попадания по пришельцам. А также хранением списка пуль, выпущенных игроком, и управлением им

* Пришелец

Класс, описывающий поведение отдельного пришельца, то есть его передвижение, отрисовка и выстрелы

* Группа пришельцев

Класс, отвечающий за совместное поведение пришельцев, движение по траектории (если крайний пришелец достигает конца экрана все пришельцы меняют направление и сдвигаются вниз), определение противника который в данный момент должен выстрелить, а также отслеживание попадания по игроку.

* Пуля

Класс, описывающий поведение пули. В зависимости от того кто создает пулю она может идти как вверх(в случае создания игроком), так и вниз(в случае создания врагом)

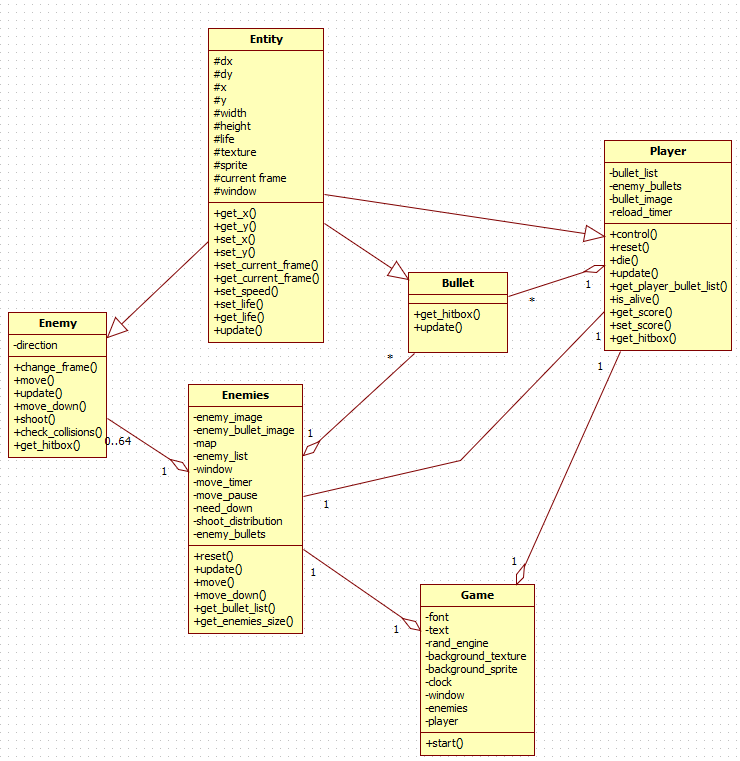


Рис2. Диаграмма классов

Заголовочные файлы

Bullet.h

#ifndef BULLET

#define BULLET

#include "entity.h"

class Bullet: public Entity{

public:

Bullet(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window, float i\_speed);

void *update*(float time);

IntRect get\_hitbox();

};

#endif // BULLET

Enemies.h

#ifndef ENEMIES

#define ENEMIES

#include <SFML/Graphics.hpp>

#include <list>

#include <random>

#include "enemy.h"

#include "globals.h"

using namespace std;

class Enemies{

Image enemy\_img;

Image bullet\_img;

String map;

list<Enemy\*> enemy\_list;

list<Enemy\*>::iterator iter;

RenderWindow\* window;

Clock move\_timer;

float move\_pause;

bool need\_down, below;

uniform\_int\_distribution<unsigned short> shoot\_distribution;

list<Bullet\*> enemy\_bullets;

list<Bullet\*>::iterator bullet\_iter;

public:

~Enemies();

Enemies(RenderWindow\* i\_window);

void reset();

void update(float time,std::mt19937& engine, list<Bullet\*>\* player\_bullet);

void move(std::mt19937& engine);

void move\_down();

list<Bullet\*>\* get\_bullet\_list();

int get\_enemies\_size();

bool get\_below();

};

#endif // ENEMIES

Enemy.h

#ifndef ENEMY

#define ENEMY

#include <random>

#include "bullet.h"

#include "entity.h"

class Enemy: public Entity{

enum direction {left=-1,right=1} state;

public:

Enemy(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window);

void change\_frame();

bool move(int dir);

void *update*(float time);

void move\_down();

Bullet\* shoot(Image &image);

IntRect get\_hitbox();

bool check\_collisions(IntRect player\_bullet);

};

#endif // ENEMY

Entitu.h

#ifndef ENTITY

#define ENTITY

#include <SFML/Graphics.hpp>

#include "globals.h"

using namespace sf;

class Entity {

protected:

float dx, dy, x, y, speed;//Г¤Г®ГЎГ ГўГЁГ«ГЁ ГЇГҐГ°ГҐГ¬ГҐГ­Г­ГіГѕ ГІГ Г©Г¬ГҐГ° Г¤Г«Гї ГЎГіГ¤ГіГ№ГЁГµ Г¶ГҐГ«ГҐГ©

int w, h; //ГЇГҐГ°ГҐГ¬ГҐГ­Г­Г Гї вЂњhealthвЂќ, ГµГ°Г Г­ГїГ№Г Гї Г¦ГЁГ§Г­ГЁ ГЁГЈГ°Г®ГЄГ

bool life; //ГЇГҐГ°ГҐГ¬ГҐГ­Г­Г Гї вЂњlifeвЂќ Г¦ГЁГ§Г­Гј, Г«Г®ГЈГЁГ·ГҐГ±ГЄГ Гї

Texture texture;//Г±ГґГ¬Г« ГІГҐГЄГ±ГІГіГ°Г

Sprite sprite;//Г±ГґГ¬Г« Г±ГЇГ°Г Г©ГІ

int CurrentFrame;//ГµГ°Г Г­ГЁГІ ГІГҐГЄГіГ№ГЁГ© ГЄГ Г¤Г°

RenderWindow\* window;

public:

Entity(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window);

float get\_x();

float get\_y();

void set\_y(float tmp\_y);

void set\_x(float tmp\_x);

void set\_current\_frame(int cur\_frame);

int get\_current\_frame();

void set\_speed(float v);

void set\_life(bool tmp\_life);

bool get\_life();

virtual void *update*(float time)=0;

};

#endif // ENTITY

Game.h

#ifndef GAME

#define GAME

#include <iostream>

#include <SFML/Graphics.hpp>

#include <random>

using namespace sf;

#include "globals.h"

#include "player.h"

#include "enemies.h"

class Game{

private:

Font font;

Text\* text;

std::mt19937 engine;

Texture background\_texture;

Sprite background\_sprite;

Clock clock;

RenderWindow\* window;

Enemies\* enemies;

Player\* player;

public:

Game(RenderWindow\* wind);

void start();

};

#endif // GAME

Globals.h

#ifndef GLOBALS

#define GLOBALS

#include <iostream>

using namespace std;

//Лень цыферки запоминать

//Расположение файлов

const string PLAYER\_BULLET = "Resources/Images/PlayerBullet.png";

const string PLAYER\_IMG="Resources/Images/Player.png";

const string BACKGROUND="Resources/Images/Background.png";

const string ENEMY\_IMG = "Resources/Images/Enemy0.png";

const string ENEMY\_BULLET="Resources/Images/EnemyBullet.png";

const string TIMES\_NEW\_ROMAN="Resources/TimesNewRoman.ttf";

//Константы окна

const unsigned short SCREEN\_HEIGHT = 180;

const unsigned short SCREEN\_WIDTH = 320;

const unsigned char SCREEN\_RESIZE = 4;

//Константы для игрока

const unsigned short BASE\_SIZE=16;//Размер спрайта(длинна и ширина)

const unsigned int RELOAD\_TIME=200;

const unsigned int ENEMY\_VALUE=5;

//Константы врагов

const unsigned int MOVE\_TIME=705;

const unsigned char STEP=1;

const float ENEMY\_BULLET\_SPEED = 0.0625;

const unsigned int ENEMY\_SHOOT\_RATE =255;

const unsigned char ENEMY\_SPEED\_INCREASE=11;

#endif // GLOBALS

Player.h

#ifndef PLAYER

#define PLAYER

#include <SFML/Graphics.hpp>

#include "globals.h"

#include "bullet.h"

#include <list>

#include "entity.h"

using namespace sf;

using namespace std;

class Player:public Entity{

private:

unsigned int score;

enum { left, right, stay} state;

list<Bullet\*> bullet\_list;

list<Bullet\*>\* enemy\_bullets;

list<Bullet\*>::iterator iter;

Image bullet\_img;

Clock reload\_timer;

public:

Player(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window, list<Bullet\*>\* enemy\_bul);

~Player();

void control();

void reset(bool win);

void die();

void *update*(float time);

list<Bullet\*>\* get\_player\_bullet\_list();

IntRect get\_hitbox();

bool is\_alive();

string get\_score();

void set\_score(unsigned int sc);

};

#endif // PLAYER

Исполнительные файлы

Bullet.cpp

#include "bullet.h"

Bullet::Bullet(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window, float i\_speed) :Entity(image, X,Y, W, H, i\_window){

speed= i\_speed;

}

void Bullet::*update*(float time){

if (life==true){

dy=speed;

y +=dy\*time;

sprite.setPosition(x,y);

window->draw(sprite);

}

}

IntRect Bullet::get\_hitbox(){

return sf::IntRect(x + 0.375f \* BASE\_SIZE, y + 0.375f \* BASE\_SIZE, 0.25f \* BASE\_SIZE, 0.25f \* BASE\_SIZE);

}

Enemies.cpp

#include "enemies.h"

Enemies::~Enemies(){

while(!enemy\_list.empty()){

iter= enemy\_list.begin();

enemy\_list.erase(iter);

}

while(!enemy\_bullets.empty()){

bullet\_iter=enemy\_bullets.begin();

enemy\_bullets.erase(bullet\_iter);

}

}

Enemies::Enemies(RenderWindow\* i\_window){

need\_down=false;

move\_timer.restart();

window=i\_window;

enemy\_img.loadFromFile(ENEMY\_IMG);

bullet\_img.loadFromFile(ENEMY\_BULLET);

map = "0000000000000000\n0000000000000000\n0000000000000000\n0000000000000000";

reset();

}

void Enemies::reset(){

below=false;

unsigned char enemy\_x = 0;

unsigned char enemy\_y = 0;

while(!enemy\_list.empty()){

iter= enemy\_list.begin();

enemy\_list.erase(iter);

}

while(!enemy\_bullets.empty()){

bullet\_iter=enemy\_bullets.begin();

enemy\_bullets.erase(bullet\_iter);

}

shoot\_distribution = uniform\_int\_distribution<unsigned short>(0,ENEMY\_SHOOT\_RATE);

for(char map\_part : map){

enemy\_x++;

switch(map\_part){

case '\n':

{

enemy\_x=0;

enemy\_y++;

break;

}

case '0':

{

Enemy\* tmp\_enemy = new Enemy(enemy\_img,BASE\_SIZE\*(1+enemy\_x),BASE\_SIZE\*(1+enemy\_y),BASE\_SIZE,BASE\_SIZE,window);

if ((enemy\_x+enemy\_y)%2){

tmp\_enemy->change\_frame();

}

enemy\_list.push\_back(tmp\_enemy);

break;

}

}

}

}

void Enemies::update(float time,std::mt19937& engine, list<Bullet\*>\* player\_bullet){

move(engine);

if (!enemy\_bullets.empty()){

bullet\_iter=enemy\_bullets.begin();

while(bullet\_iter != enemy\_bullets.end()){

(\*bullet\_iter)->update(time);

if ((\*bullet\_iter)->get\_y()>SCREEN\_HEIGHT){

enemy\_bullets.erase(bullet\_iter);

}

bullet\_iter++;

}

}

if(!enemy\_list.empty()){

iter = enemy\_list.begin();

while(iter != enemy\_list.end()){

(\*iter)->update(time);

if ((\*iter)->get\_y()>=SCREEN\_HEIGHT-3\*BASE\_SIZE/2){

below=true;

}

for(bullet\_iter=player\_bullet->begin();bullet\_iter!=player\_bullet->end();bullet\_iter++){

if ((\*iter)->check\_collisions((\*bullet\_iter)->get\_hitbox())){

enemy\_list.erase(iter);

(\*bullet\_iter)->set\_life(false);

break;

}

}

iter++;

}

}

}

void Enemies::move(std::mt19937& engine){

if (move\_timer.getElapsedTime().asMilliseconds()>MOVE\_TIME-ENEMY\_SPEED\_INCREASE\*(64-enemy\_list.size())){

if(!need\_down){

if(!enemy\_list.empty()){

iter = enemy\_list.begin();

while(iter != enemy\_list.end()){

if ((\*iter)->move(1)){

need\_down=true;

}

if (shoot\_distribution(engine)==0){enemy\_bullets.push\_back((\*iter)->shoot(bullet\_img));}

iter++;

}

}

move\_timer.restart();

}else{

move\_down();

need\_down=false;

}

}

}

void Enemies::move\_down(){

if(!enemy\_list.empty()){

iter = enemy\_list.begin();

while(iter != enemy\_list.end()){

(\*iter)->move\_down();

(\*iter)->move(-1);

iter++;

}

}

}

list<Bullet\*>\* Enemies::get\_bullet\_list(){

return &enemy\_bullets;

}

int Enemies::get\_enemies\_size(){

return enemy\_list.size();

}

bool Enemies::get\_below()

{

return below;

}

Enemy.cpp

#include"enemy.h"

Enemy::Enemy(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window) :Entity(image, X,

Y, W, H, i\_window){

sprite.setTextureRect(IntRect(CurrentFrame\*w,0,w,h));

state = right;

sprite.setPosition(x, y);

window->draw(sprite);

set\_speed(STEP);

}

void Enemy::change\_frame(){

if(CurrentFrame==0){

set\_current\_frame(1);

}else{

set\_current\_frame(0);

}

}

bool Enemy::move(int dir){

change\_frame();

bool return\_value=0;

if (state==right){

return\_value = (x + BASE\_SIZE + STEP >= SCREEN\_WIDTH);

}else{

return\_value= (x - STEP <=0);

}

x+=speed\*state\*dir;

return return\_value;

}

void Enemy::*update*(float time){

sprite.setTextureRect(IntRect(CurrentFrame\*w,0,w,h));

sprite.setPosition(x, y);

window->draw(sprite);

}

void Enemy::move\_down(){

if (state==left){

state = right;

}else{

state = left;

}

y+=BASE\_SIZE;

}

Bullet\* Enemy::shoot(Image &image){

Bullet\* tmp\_bullet = new Bullet(image,x,y,BASE\_SIZE,BASE\_SIZE,window,ENEMY\_BULLET\_SPEED);

return tmp\_bullet;

}

IntRect Enemy::get\_hitbox()

{

return IntRect(x + 0.25f \* BASE\_SIZE, y + 0.25f \* BASE\_SIZE, 0.5f \* BASE\_SIZE, 0.5f \* BASE\_SIZE);

}

bool Enemy::check\_collisions(IntRect player\_bullet){

if (get\_hitbox().intersects(player\_bullet)){

return true;

}else{

return false;

}

}

Entity.cpp

#include "entity.h"

Entity::Entity(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window){

x = X; y = Y; //пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅ

w = W; h = H;

window = i\_window;

dx = 0; dy = 0;

speed = 0;

life = true; //пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅ, пїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅ

texture.loadFromImage(image); //пїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ

sprite.setTexture(texture); //пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅ пїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅпїЅ

}

float Entity::get\_x(){

return x;

}

float Entity::get\_y(){

return y;

}

void Entity::set\_y(float tmp\_y){

y=tmp\_y;

}

void Entity::set\_x(float tmp\_x){

x=tmp\_x;

}

void Entity::set\_current\_frame(int cur\_frame){

CurrentFrame = cur\_frame;

}

int Entity::get\_current\_frame(){

return CurrentFrame;

}

void Entity::set\_speed(float v){

speed=v;

}

void Entity::set\_life(bool tmp\_life){

life = tmp\_life;

}

bool Entity::get\_life(){

return life;

}

Game.cpp

#include "game.h"

Game::Game(RenderWindow\* wind):window(wind){

font.loadFromFile(TIMES\_NEW\_ROMAN);

text=new Text("", font, 16);

text->setColor(Color::Blue);

engine.seed(std::time(nullptr));

background\_texture.loadFromFile(BACKGROUND);

background\_sprite.setTexture(background\_texture);

enemies=new Enemies(window);

Image player\_img;

player\_img.loadFromFile(PLAYER\_IMG);

player=new Player(player\_img, 0,SCREEN\_HEIGHT-BASE\_SIZE-1, BASE\_SIZE, BASE\_SIZE, window, enemies->get\_bullet\_list());

}

void Game::start(){

while (window->isOpen()){

window->clear(); //РћС‡РёС‰Р°РµРј СЌРєСЂР°РЅ

sf::Event event; //РџРµСЂРµРјРµРЅРЅР°СЏ РґР»СЏ СЃРѕР±С‹С‚РёСЏ

while (window->pollEvent(event)) //РћРїСЂРѕСЃ СЃРѕР±С‹С‚РёР№

{

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

window->close();//Р—Р°РєСЂС‹РІР°РµРј РѕРєРЅРѕ, РµСЃР»Рё СЃРѕР±С‹С‚РёРµ вЂњClosedвЂќ

}

if(Keyboard::isKeyPressed(Keyboard::Escape)){

window->close();

}

if(Keyboard::isKeyPressed(Keyboard::Space)){

enemies->reset();

player->reset(false);

}else{

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

clock.restart();

time = time / 800;

if (player->get\_life()&& !(enemies->get\_below())){

if(enemies->get\_enemies\_size()==0){

player->reset(true);

enemies->reset();

}

window->draw(background\_sprite);

enemies->update(time, engine,player->get\_player\_bullet\_list());

player->*update*(time);

text->setPosition(0, -5);

text->setString("score:"+player->get\_score());

window->draw(\*text);

}else{

text->setPosition(0, -5);

text->setString("score:"+player->get\_score());

text->setPosition(60, 60);

if (!player->get\_life()){

text->setString("Game Over(you were killed)");

}else{

text->setString("Game Over(enemies reached you)");

}

window->draw(\*text);

text->setPosition(110, 77);

text->setString("Your score:"+player->get\_score());

window->draw(\*text);

text->setPosition(90, 93);

text->setString("Press 'space' to start");

window->draw(\*text);

}

}

window->display(); //РћС‚РѕР±СЂР°Р¶Р°РµРј РєСЂСѓРі РЅР° СЌРєСЂР°РЅ

}

}

Main.cpp

#include "game.h"

int main()

{

RenderWindow window(VideoMode(SCREEN\_RESIZE \* SCREEN\_WIDTH, SCREEN\_RESIZE \* SCREEN\_HEIGHT), "Space Invaders", Style::Close);

window.setView(View(FloatRect(0, 0, SCREEN\_WIDTH, SCREEN\_HEIGHT)));

Game game(&window);

game.start();

}

Player.cpp

#include "player.h"

Player::Player(Image &image, float X, float Y, int W, int H, RenderWindow\* i\_window, list<Bullet\*>\* enemy\_bul) :Entity(image, X,

Y, W, H, i\_window),enemy\_bullets(enemy\_bul){

score=0;

bullet\_list.clear();

state = stay;

sprite.setTextureRect(IntRect(0, 0, w, h));

bullet\_img.loadFromFile("Resources/Images/PlayerBullet.png");

}

Player::~Player(){

while(!bullet\_list.empty()){

iter= bullet\_list.begin();

bullet\_list.erase(iter);

}

}

void Player::control(){

if (Keyboard::isKeyPressed(Keyboard::Left)) {

if(!(x+BASE\_SIZE/2<0)){

state = left;

speed = 0.1;}

}

if (Keyboard::isKeyPressed(Keyboard::Right) ) {

if(!(x+BASE\_SIZE/2>SCREEN\_WIDTH)){

state = right;

speed = 0.1;

}

}

if (Keyboard::isKeyPressed(Keyboard::Up)) {

if (reload\_timer.getElapsedTime().asMilliseconds() > RELOAD\_TIME){

Bullet\* tmp\_bullet= new Bullet(bullet\_img,x,y-BASE\_SIZE,BASE\_SIZE,BASE\_SIZE,window,-0.2);

bullet\_list.push\_back(tmp\_bullet);

reload\_timer.restart();

}

}

}

void Player::reset(bool win){

x=0;

y=SCREEN\_HEIGHT-BASE\_SIZE-1;

if (!win){

score=0;

life=true;

}

while(!bullet\_list.empty()){

iter= bullet\_list.begin();

bullet\_list.erase(iter);

}

}

void Player::die(){

life=false;

}

void Player::*update*(float time) //РјРµС‚РѕРґ "РѕР¶РёРІР»РµРЅРёВ¤/РѕР±РЅРѕРІР»РµРЅРёВ¤" РѕР±СЉРµРєС‚Р° РєР»Р°СЃСЃР°.

{

if (life) {//РїСЂРѕРІРµСЂВ¤РµРј, Р¶РёРІ Р»Рё РіРµСЂРѕР№

control();//С„СѓРЅРєС†РёВ¤ СѓРїСЂР°РІР»РµРЅРёВ¤ РїРµСЂСЃРѕРЅР°Р¶РµРј

switch (state)//РґРµР»Р°СЋС‚СЃВ¤ СЂР°Р·Р»РёС‡РЅС‹Рµ РґРµР№СЃС‚РІРёВ¤ РІ Р·Р°РІРёСЃРёРјРѕСЃС‚Рё РѕС‚ СЃРѕСЃС‚РѕВ¤РЅРёВ¤

{

case right:{

dx = speed;

break;

}

case left:{

dx = -speed;

break;

}

case stay:{//СЃС‚РѕРёРј

dy = speed;

dx = speed;

break;

}

}

x += dx\*time; //РґРІРёР¶РµРЅРёРµ РїРѕ РЈXР¤

speed = 0; //РѕР±РЅСѓР»В¤РµРј СЃРєРѕСЂРѕСЃС‚СЊ, С‡С‚РѕР±С‹ РїРµСЂСЃРѕРЅР°Р¶ РѕСЃС‚Р°РЅРѕРІРёР»СЃВ¤..

sprite.setPosition(x, y);

window->draw(sprite);

iter = bullet\_list.begin();

while (iter != bullet\_list.end()){

(\*iter)->update(time);

if((\*iter)->get\_y()<-BASE\_SIZE || (\*iter)->get\_life()==false){

if ((\*iter)->get\_life()==false)

score+=ENEMY\_VALUE;

bullet\_list.erase(iter);

}

\*iter++;

}

for (iter=enemy\_bullets->begin();iter!=enemy\_bullets->end();iter++){

if (get\_hitbox().intersects((\*iter)->get\_hitbox())){

die();

}

}

}

}

list<Bullet\*>\* Player::get\_player\_bullet\_list(){

return &bullet\_list;

}

IntRect Player::get\_hitbox()

{

return IntRect(x + 0.125f \* BASE\_SIZE, y + 0.125f \* BASE\_SIZE, 0.75f \* BASE\_SIZE, 0.75f \* BASE\_SIZE);

}

bool Player::is\_alive(){

return life;

}

string Player::get\_score(){

string str = to\_string(score);

return str;

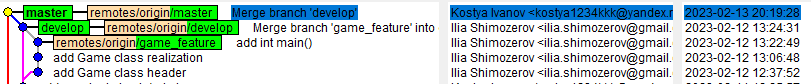
}

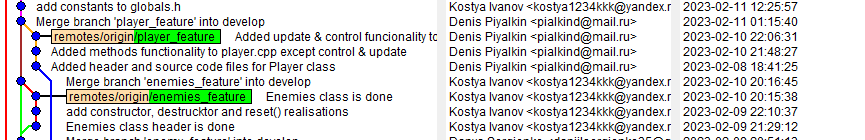
void Player::set\_score(unsigned int sc){

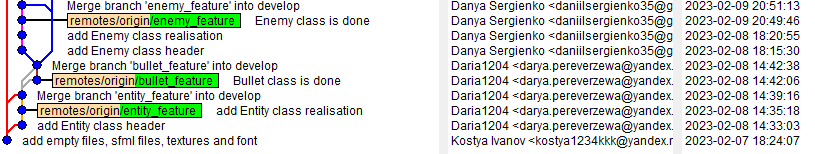
score=sc;

}

История проекта Github:







(Иванов Константин)

Add empty files, sfml files, textures and font

В проект добавлены пустые файлы под классы, текстуры, шрифт, и файлы необходимые для sfml

(Переверзева Дарья)

add Entity class header

добавлен готовый заголовочный файл класса Entity

add Entity class realization

добавлены реализации к методам заголовочного файла

Merge branch 'entity\_feature' into develop

Вливание ветки entity\_feature в develop

Bullet class is done

Полностью реализован класс пуля(как заголовочный файл так и файл реализации)

Merge branch 'bullet\_feature' into develop

Вливание ветки bullet\_feature в develop

(Сергиенко Даниил)

add Enemy class header

добавлен готовый заголовочный файл класса Enemy

add Enemy class realization

добавлены реализации к методам заголовочного файла

Enemy class is done

Добавлены методы shoot(), get\_hitbox(), check\_collisions() а также их реализации

Merge branch 'enemy\_feature' into develop

Вливание ветки enemy\_feature в develop

(Иванов Константин)

Enemies class header is done

добавлен готовый заголовочный файл класса Enemies

add constructor, destructor and reset() realisations

добавлены реализации конструктора, деструктора и метода reset()

Enemies class is done

Добавлены остальные реализации методов класса Enemies

Merge branch 'enemies\_feature' into develop

Вливание ветки enemies\_feature в develop

(Пиялкин Денис)

Added header and source code files for Player class

Добавлен готовый заголовочный файл класса Player и заглушки под методы

Added methods functionality to player.cpp except control & update

Добавлены реализация всем методам кроме control() и update()

Added update & control funcionality to player.cpp

Добавлена реализация методам control() и update()

Merge branch 'player\_feature' into develop

Влияние ветки player\_feature в develop

(Иванов Константин)

add constants to globals.h

добавлен заголовочный файл с константами

(Шимозеров Илья)

add Game class header

добавлен готовый заголовочный файл класса Game

add Game class realization

добавлена реализация методов

add int main()

добавлена функция main()

Merge branch 'game\_feature' into develop

(Иванов Константин)

Merge branch 'develop'

Вливание ветки develop в master

**Вывод:** В ходе выполнения командного задания в рамках курсового проекта использовалась среда разработки Qt язык C++ с использованием библиотеки SFML. Применялась система контроля версий git. Реализованы все прецеденты. В ходе тестирования игры зависания или сбои не были обнаружены. Для классов Player и Enemies, хранящих в себе списки пуль, была реализована очистка памяти в их деструкторах. Неиспользуемые переменные в ходе разработки старались удалять. В алгоритмы методов старались не добавлять не нужных циклов, массивов и др. К отчету приложены диаграмма вариантов использования, диаграммы деятельности, диаграммы последовательности и диаграмма классов. Все они, за исключением диаграммы вариантов использования, находятся в пошаговом описании процесса разработки. Мы считаем что наша команда достигла поставленных целей.