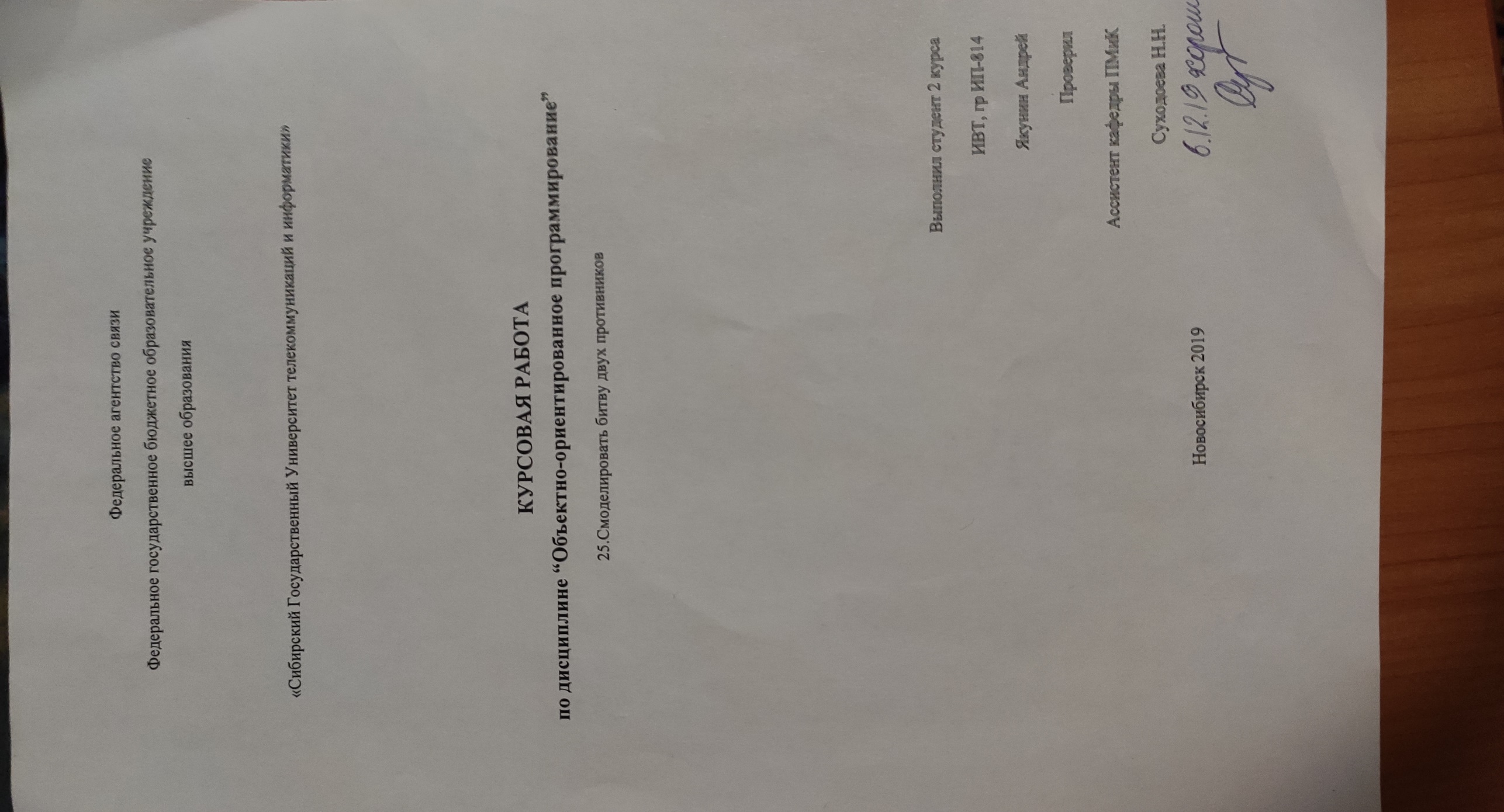
****

**1. Вариант курсовой работы №11**

Написать программу, используя объектно-ориентированный подход, реализующий игру «крестики-нолики» в графическом режиме.

**Постановка задачи:**

Игрокам предоставляется поле, каждый игрок по очереди ставит крестик или нолик в одной из клетки поля. Победителем считается игрок, собравший победную комбинацию. При победе выигрышная комбинация помечается цветом.

**Реализация программного кода:**

Для реализации Программы была использована свободная кроссплатформенная мультимедийная библиотека SFML.

**Описание алгоритма основной программы**

В основной программе идет объявление основных объектов и задаются настройки для окна, также объявляются необходимые для работы программы переменные. Далее в цикле, который работает пока не будет закрыто окно, в данном цикле проверяется нажатие клавиши мыши игроком, определение нажатия в определенную клетку, отрисовка всех объектов и также проверяется победная комбинация.

**Описание иерархии и методом класса**

Figure

BGimage

Krestik

**Технологии ООП использованные в работе**

Инкапсуляция, наследование, полиморфизм, конструкторы, перегрузка конструкторов, списки иницилизации, классы, виртуальные функции, массивы указателей на объекты.

**Main:**

В главном модуле были объявленны объекты **Krestik** и **BGimage**, затем был создан массив класса **Krestik**, в котором хранятся адреса объектов класса **Krestik**. В основном цикле **while** **(window.isOpen())**, описана основная логика игры, проверяется нажатие клавиши мыши игроком, отслеживаются координаты мыши и клетки, в область которой было произведено нажатие и так же идет взаимодействие с внешними классами **Krestik** и **BGimage.**

#include <iostream>

#include <SFML/Graphics.hpp>

#include <ctime>

#include "Tank.h"

#include "Player.h"

#include "Bullet.h"

using namespace std;

using namespace sf;

float x,y;

int T = 0, T\_enemy = 0;

int bullet\_dir, bullet\_enemy\_dir, last , last\_enemy, i = 0, e = 0, q = 5;

int main()

{

RenderWindow window(sf::VideoMode(1366, 768), "TANCHIKI\_simple", Style::Fullscreen);

Clock clock;

srand(time(0));

float time;

bool T\_on = false, T\_on\_enemy = false;

Bullet \*BULLETS[9];

Bullet obj\_bullet\_0("bullet.png", 78, 128, 0.15, 0.15);

BULLETS[0] = &obj\_bullet\_0;

Bullet obj\_bullet\_1("bullet.png", 78, 128, 0.15, 0.15);

BULLETS[1] = &obj\_bullet\_1;

Bullet obj\_bullet\_2("bullet.png", 78, 128, 0.15, 0.15);

BULLETS[2] = &obj\_bullet\_2;

Bullet obj\_bullet\_3("bullet.png", 78, 128, 0.15, 0.15);

BULLETS[3] = &obj\_bullet\_3;

Bullet obj\_bullet\_4("bullet.png", 78, 128, 0.15, 0.15);

BULLETS[4] = &obj\_bullet\_4;

Bullet obj\_bullet\_enemy\_0("bullet\_enemy.png", 78, 128, 0.15, 0.15);

BULLETS[5] = &obj\_bullet\_enemy\_0;

Bullet obj\_bullet\_enemy\_1("bullet\_enemy.png", 78, 128, 0.15, 0.15);

BULLETS[6] = &obj\_bullet\_enemy\_1;

Bullet obj\_bullet\_enemy\_2("bullet\_enemy.png", 78, 128, 0.15, 0.15);

BULLETS[7] = &obj\_bullet\_enemy\_2;

Bullet obj\_bullet\_enemy\_3("bullet\_enemy.png", 78, 128, 0.15, 0.15);

BULLETS[8] = &obj\_bullet\_enemy\_3;

Bullet obj\_bullet\_enemy\_4("bullet\_enemy.png", 78, 128, 0.15, 0.15);

BULLETS[9] = &obj\_bullet\_enemy\_4;

Player obj\_enemy("character.png", 300, 200, 78, 128, 0.5, 0.5);

Player obj\_player("character\_enemy.png");

Image map\_image;

map\_image.loadFromFile("image/map/pesok.jpg");

Texture map;

map.loadFromImage(map\_image);

Sprite s\_map;

s\_map.setTexture(map);

window.setVerticalSyncEnabled(true);

time = clock.getElapsedTime().asMilliseconds();

clock.restart();

while (window.isOpen())

{

time = time / 800;

Event event;

while (window.pollEvent(event))

{

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

window.close();

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

if (event.key.code == sf::Keyboard::Escape)

window.close();

}

//////////////////////////////////////////Enemy - upravlenie

if (Keyboard::isKeyPressed(Keyboard::Left) and obj\_enemy.life == true)

{

obj\_enemy.dir = 0;

obj\_enemy.Move();

bullet\_enemy\_dir = 0;

}

else if (Keyboard::isKeyPressed(Keyboard::Right) and obj\_enemy.life == true)

{

obj\_enemy.dir = 1;

obj\_enemy.Move();

bullet\_enemy\_dir = 1;

}

else if (Keyboard::isKeyPressed(Keyboard::Up) and obj\_enemy.life == true)

{

obj\_enemy.dir = 2;

obj\_enemy.Move();

bullet\_enemy\_dir = 2;

}

else if (Keyboard::isKeyPressed(Keyboard::Down) and obj\_enemy.life == true)

{

obj\_enemy.dir = 3;

obj\_enemy.Move();

bullet\_enemy\_dir = 3;

}

//////////////////////////////////////////Enemy - upravlenie

//////////////////////////////////////////Player - upravlenie

if (Keyboard::isKeyPressed(Keyboard::A) and obj\_player.life == true)

{

obj\_player.dir = 0;

obj\_player.Move();

bullet\_dir = 0;

}

else if (Keyboard::isKeyPressed(Keyboard::D) and obj\_player.life == true)

{

obj\_player.dir = 1;

obj\_player.Move();

bullet\_dir = 1;

}

else if (Keyboard::isKeyPressed(Keyboard::W) and obj\_player.life == true)

{

obj\_player.dir = 2;

obj\_player.Move();

bullet\_dir = 2;

}

else if (Keyboard::isKeyPressed(Keyboard::S) and obj\_player.life == true)

{

obj\_player.dir = 3;

obj\_player.Move();

bullet\_dir = 3;

}

//////////////////////////////////////////Player - upravlenie

//////////////////////////////////////////BULLET

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

{

BULLETS[e]->dir = bullet\_dir;

if (last == 0 and T == 0 and BULLETS[e]->shoot == false and obj\_player.life == true)

{

x = obj\_player.sprite.getPosition().x + 10;

y = obj\_player.sprite.getPosition().y + 14;

BULLETS[e]->sprite.setPosition(x,y);

BULLETS[e]->shoot == true;

e++;

T = 0;

T\_on = true;

}

if (last == 1 and T == 0 and BULLETS[e]->shoot == false and obj\_player.life == true)

{

x = obj\_player.sprite.getPosition().x + 30;

y = obj\_player.sprite.getPosition().y + 14;

BULLETS[e]->sprite.setPosition(x,y);

BULLETS[e]->shoot == true;

e++;

T = 0;

T\_on = true;

}

if (last == 2 and T == 0 and BULLETS[e]->shoot == false and obj\_player.life == true)

{

x = obj\_player.sprite.getPosition().x + 14;

y = obj\_player.sprite.getPosition().y + 10;

BULLETS[e]->sprite.setPosition(x,y);

BULLETS[e]->shoot == true;

e++;

T = 0;

T\_on = true;

}

if (last == 3 and T == 0 and BULLETS[e]->shoot == false and obj\_player.life == true)

{

x = obj\_player.sprite.getPosition().x + 14;

y = obj\_player.sprite.getPosition().y + 30;

BULLETS[e]->sprite.setPosition(x,y);

BULLETS[e]->shoot == true;

e++;

T = 0;

T\_on = true;

}

if (e > 4) e = 0;

}

//////////////////////////////////////////BULLET

//////////////////////////////////////////BULLET\_enemy

if (Keyboard::isKeyPressed(Keyboard::Numpad0))

{

BULLETS[q]->dir = bullet\_enemy\_dir;

if (last\_enemy == 0 and T\_enemy == 0 and BULLETS[e]->shoot == false and obj\_enemy.life == true)

{

x = obj\_enemy.sprite.getPosition().x + 10;

y = obj\_enemy.sprite.getPosition().y + 14;

BULLETS[q]->sprite.setPosition(x,y);

BULLETS[q]->shoot == true;

T\_enemy = 0;

T\_on\_enemy = true;

q++;

}

if (last\_enemy == 1 and T\_enemy == 0 and BULLETS[e]->shoot == false and obj\_enemy.life == true)

{

x = obj\_enemy.sprite.getPosition().x + 30;

y = obj\_enemy.sprite.getPosition().y + 14;

BULLETS[q]->sprite.setPosition(x,y);

BULLETS[q]->shoot == true;

T\_enemy = 0;

T\_on\_enemy = true;

q++;

}

if (last\_enemy == 2 and T\_enemy == 0 and BULLETS[e]->shoot == false and obj\_enemy.life == true)

{

x = obj\_enemy.sprite.getPosition().x + 14;

y = obj\_enemy.sprite.getPosition().y + 10;

BULLETS[q]->sprite.setPosition(x,y);

BULLETS[q]->shoot == true;

T\_enemy = 0;

T\_on\_enemy = true;

q++;

}

if (last\_enemy == 3 and T\_enemy == 0 and BULLETS[e]->shoot == false and obj\_enemy.life == true)

{

x = obj\_enemy.sprite.getPosition().x + 14;

y = obj\_enemy.sprite.getPosition().y + 30;

BULLETS[q]->sprite.setPosition(x,y);

BULLETS[q]->shoot == true;

T\_enemy = 0;

T\_on\_enemy = true;

q++;

}

if (q > 9) q = 5;

}

//////////////////////////////////////////BULLET\_enemy

for (int w = 0; w < 9; w++) BULLETS[w]->Move();

if (T\_on\_enemy == true)

{

T\_enemy++;

if (T\_enemy > 30)

{

T\_enemy = 0;

T\_on\_enemy = false;

}

}

if (T\_on == true)

{

T++;

if (T > 30)

{

T = 0;

T\_on = false;

}

}

for (int en = 0; en < 4; en++) if (BULLETS[en]->sprite.getGlobalBounds().intersects(obj\_enemy.sprite.getGlobalBounds())) obj\_enemy.life = false;

for (int en = 5; en < 9; en++) if (BULLETS[en]->sprite.getGlobalBounds().intersects(obj\_player.sprite.getGlobalBounds())) obj\_player.life = false;

if (obj\_player.life == false) obj\_player.sprite.setColor(Color::Red);

if (obj\_enemy.life == false) obj\_enemy.sprite.setColor(Color::Red);

if (Keyboard::isKeyPressed(Keyboard::R))

{

obj\_player.life = true;

obj\_player.sprite.setColor(Color::White);

obj\_enemy.sprite.setColor(Color::White);

obj\_enemy.life = true;

}

window.clear();

window.draw(s\_map);

window.draw(obj\_player.sprite);

window.draw(obj\_enemy.sprite);

for (int o = 0; o < 9; o++) window.draw(BULLETS[o]->sprite);

window.display();

}

return 0;

}

**Tank.h:**

Абстрактный класс имеющий чистую виртуальную функцию и дающий классам наследникам общие переменные.

#ifndef TANK\_H

#define TANK\_H

#include <iostream>

#include <SFML/Graphics.hpp>

#include <ctime>

using namespace std;

using namespace sf;

class Tank

{

public:

float w, h, x, y, speed, w\_scale, h\_scale;

int dir;

bool life = true;

bool shoot = false;

String File;

Image image;

Texture texture;

Sprite sprite;

virtual void Move()=0;

};

#endif

**Player.h:**

Класс Наследник Tank.h который отвечает за движение танков Игрока и Противника. Имеющий перегрузку конструкторов на основе которых создаются танк Игрока и Противника.

#ifndef PLAYER\_H

#define PLAYER\_H

#include "Tank.h"

class Player : public Tank

{

public:

float w, h, x, y, speed, w\_scale, h\_scale;

Player(String F, float X, float Y, float W, float H, float W\_SCALE, float H\_SCALE);

Player(String F) : w(0.5), h(0.5)

{

File = F;

image.loadFromFile("image/character/" + File);

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setPosition(600, 300);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(21, 194, 78, 128));

sprite.setScale(w, h);

}

void Move();

protected:

};

#endif

**Player.cpp:**

В данном модуле реализована работа функции Move отвечающей за перемещение танков по игровому полю.

#include "Player.h"

Player::Player(String F, float X, float Y, float W, float H, float W\_SCALE, float H\_SCALE)

{

w\_scale = W\_SCALE;

h\_scale = H\_SCALE;

w = W;

h = H;

x = X;

y = Y;

File = F;

image.loadFromFile("image/character/" + File);

texture.loadFromImage(image);

sprite.setPosition(x, y);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(21, 194, w, h));

sprite.setScale(w\_scale, h\_scale);

}

void Player::Move()

{

if (dir == 0)

{

sprite.move(-5,0);

sprite.setTextureRect(IntRect(138,255,128,78));

}

if (dir == 1)

{

sprite.move(5,0);

sprite.setTextureRect(IntRect(12,58,128,78));

}

if ( dir == 2)

{

sprite.move(0,-5);

sprite.setTextureRect(IntRect(21,194,78,128));

}

if (dir == 3)

{

sprite.move(0,5);

sprite.setTextureRect(IntRect(188,32,78,128));

}

}

**Bullet.h:**

Класс Наследник Tank.h который отвечает за движение пуль Игрока и Противника. Имеющий конструктор на основе которого создаются снаряды, наносящие урон Игроку и Противнику.

#ifndef BULLET\_H

#define BULLET\_H

#include "Tank.h"

class Bullet : public Tank

{

public:

Bullet(String F, float W, float H, float W\_SCALE, float H\_SCALE)

{

w\_scale = W\_SCALE;

h\_scale = H\_SCALE;

w = W;

h = H;

File = F;

image.loadFromFile("image/character/" + File);

texture.loadFromImage(image);

sprite.setTextureRect(IntRect(12,58,128,78));

sprite.setTexture(texture);

sprite.setScale(w\_scale, h\_scale);

}

void Move();

};

#endif

**Bullet.cpp:**

В данном модуле реализована работа функции Move отвечающей за перемещение танков по игровому полю.

#include "Bullet.h"

void Bullet::Move()

{

if (dir == 0)

{

sprite.move(-10,0);

sprite.setTextureRect(IntRect(138,255,128,78));

}

if (dir == 1)

{

sprite.move(10,0);

sprite.setTextureRect(IntRect(12,58,128,78));

}

if (dir == 2)

{

sprite.move(0,-10);

sprite.setTextureRect(IntRect(21,194,78,128));

}

if (dir == 3)

{

sprite.move(0,10);

sprite.setTextureRect(IntRect(188,32,78,128));

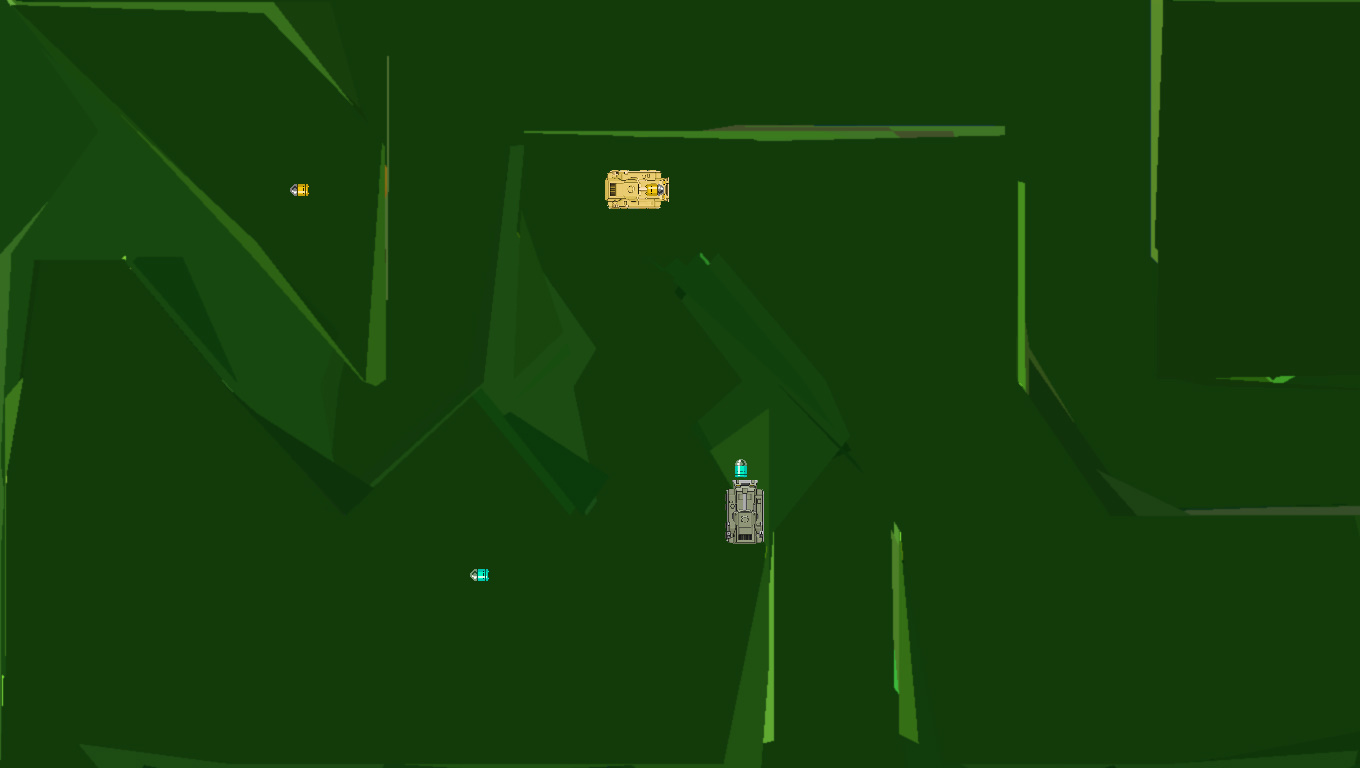
}

}

Как можно заметить в модулях Player.cpp и Bullet.cpp используется полиморфизм, позволяющий именовать функции, а также переменные одинаковыми названиями.

Реализована Инкапсуляция, благодаря которой каждая Важная часть кода (например: Классы, Функции реализованные относительно классов) находятся в различных частях проекта и позволяют упрощенно воспринимать и вносить изменения в рабочий код программы.

**Скриншоты работы программы:**



**Заключение**

В написанной нами курсовой работе нами использовались различные методы ООП, например списки инициализации в модуле Player.h. Так же был реализован полиморфизм для именования функций и переменных одинаковыми именами. Реализована Инкапсуляция, благодаря которой каждая Важная часть кода (например: Классы, Функции реализованные относительно классов) находятся в различных частях проекта и позволяют упрощенно воспринимать код и скрывать от пользователей часть кода в которую необходимо вносить изменения.Так же был сделан абстрактный класс являющийся отцовским для удобного наследования и объявления переменных. Была реализована Чистая виртуальная функция наследуемая двумя различными Кдочерними классами. Использование методов ООП облегчило задачу создания курсовой работы, помогло создать более устойчивый и удобный для чтения и изменения код.

**Листинг**

1. Постановка задачи

1. Реализация программного кода

1. Описание алгоритма основной программы

1. Описание иерархии и методов класса

2. Технологии ООП использованные в работе

3. Main

11. Tank.h

12. Player.h

13. Player.cpp