Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Тверской государственный университет»

Математический факультет

Специальность «Компьютерная безопасность»

Кафедра компьютерной безопасности и математических методов управления

**Создание игрового проекта с использованием мультимедийной библиотеки. Игра «Танки»**

курсовая работа по дисциплине

**«Языки программирования»**

Автор:

Кокорин Д.А., второй курс, группа М-24

Научный руководитель:

ст. пр. кафедры

Тишина Е.В.

Тверь 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc11605763)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc11605764)

[Инструментарий программиста 4](#_Toc11605765)

[СКВ Git 6](#_Toc11605766)

[Мультимедийная библиотека SFML 7](#_Toc11605767)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 8](#_Toc11605768)

[Анализ задания 8](#_Toc11605769)

[Разработка алгоритма 9](#_Toc11605770)

[Проектирование системы классов 10](#_Toc11605771)

[ЗАКЛЮЧЕНИЕ 20](#_Toc11605772)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc11605773)

[ПРИЛОЖЕНИЕ А. ДИАГРАММА КЛАССОВ 22](#_Toc11605774)

[ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД ТЕСТОВОГО ПРИЛОЖЕНИЯ 22](#_Toc11605775)

ВВЕДЕНИЕ

В данной работе описан один из способов применения мультимедийной библиотеки в области программирования для разработки видеоигр на языке C++. Материал включает в себя такие особенности языка, как ООП, наследование классов, их композиция, а также использование графического интерфейса (GUI) с помощью приведённой выше библиотеки SFML.

Актуальность данной работы обусловливается необходимостью пользователей сегодня в программном обеспечении, сохраняющем простоту использования благодаря его реализации на графической основе.

Данный материал предназначен для начинающих разработчиков, освоивших базовую часть языка C++, и направлен на ознакомление с мультимедийной библиотекой SFML.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Инструментарий программиста

Текстовый редактор — компьютерная программа, предназначенная для создания и изменения текстовых данных в общем и текстовых файлов, в частности. В программировании используется для изменения исходного кода программ, поскольку во многих редакторах присутствуют такие особенности, как распознание языка, автозаполнение слов, автоматическая вставка отступов и др.

Компилятор — программа или техническое средство, выполняющее *компиляцию*, т.е. сборку программы, включающую её «перевод» на язык, который понимает и может воспроизвести вычислительная техника, иначе говоря, **машинный код**.

Интерпретатор – программа, включающий интерпретатор, т.е. построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от компиляции, где весь текст программы, перед запуском, анализируется и транслируется в машинный или байт-код, без её выполнения).

Отладчик (в жаргоне программистов «дебаггер») — компьютерная программа, предназначенная для поиска ошибок программе. Отладчик позволяет выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и т.д.

Всё вышеперечисленное может заменить IDE (сокр. от англ. Integrated development environment) или ИСР (интегрированная среда разработки) — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Примеры IDE:

* Visual Studio (Microsoft)
* CLion (Jetbrains)
* XCode (Apple)

СКВ (система управления версиями, от англ. Version Control System, VCS) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

СКВ Git

Git — распределённая (с особенностью локализации данных проекта каждого разработчика) система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, для того, чтобы другие разработчики могли вносить свой вклад в ядро Linux. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами.

Git стоит отдельно от других СКВ из-за подхода к работе с данными. Большинство других систем хранят информацию в виде списка изменений в файлах. Вместо этого подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Преимущества Git:

* Бесплатный и open-source. Это значит, что его можно бесплатно скачать и вносить любые изменения в исходный код;
* Небольшой и быстрый. Он выполняет все операции локально, что увеличивает его скорость. Кроме того, Git локально сохраняет весь репозиторий в небольшой файл без потери качества данных;
* Резервное копирование. Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при использовании Git;
* Простое ветвление. В других СКВ создание веток— утомительная и трудоёмкая задача, так как весь код копируется в новую ветку. В Git управление ветками реализовано гораздо проще и эффективнее.

Мультимедийная библиотека SFML

Simple and Fast Multimedia Library (SFML) – свободная кросс-платформерная мультимедийная библиотека, направленная на обеспечение простого программного интерфейса (API) для доступа ко множеству компонентов мультимедиа на компьютере. Написана на C++, и кроме этого, доступна для использования в разработке приложений на C, C#, D, Euphoria, Go, Java, Julia, .NET, Nim, OCaml, Python, Ruby и Rust.

SFML позволяет создавать и настраивать операции ввода-вывода с окнами и элементами OpenGL. Библиотека включает в себя модули:

* System (управление временем и потоками, он является обязательным, так как все модули зависят от него);
* Window (управление окнами и взаимодействием с пользователем);
* Graphics (делает простым отображение графических примитивов и изображений);
* Audio (предоставляет интерфейс для управления звуком);
* Network (для сетевых приложений).

ПРАКТИЧЕСКАЯ ЧАСТЬ

Анализ задания

Танки (в оригинале Battle City) — компьютерная игра для игровых приставок Famicom (NES) и Game Boy. Мировая премьера – 1985 г.

Полигон действий виден сверху. Игрок должен, управляя своим танком, уничтожить все вражеские танки на уровне. Враги пытаются уничтожить штаб игрока (внизу игрового поля в виде орла) и его танк. Если противник сможет разрушить штаб или лишит игрока всех жизней — игра окончена.

Разработка алгоритма



Проектирование системы классов

В программе используется восемь классов:

* Application – «голова» программы. В ней протекает весь процесс игры. Все остальные классы служат частью этого класса;
* Tank – сущность танка, её графическая интерпретация, движение, анимация, взаимодействие с танками и картой и т.д. Базовый класс для Enemy и Player;
* Enemy – вражеские танки, цели игрока. Движение и стрельба происходят случайным образом. Наследуется от Tank;
* Player – танк игрока с возможностью выстрела по команде (нажатием Space). Наследуется от Tank;
* Base – штаб, цель врагов и объект защиты игрока от нападения;
* Bullet – боеприпасы танка. Часть класса Tank;
* Map – поле боя с препятствиями в виде кирпичных стен;
* Message – сообщение о состоянии игры на экране.

Описания и реализации классов:

|  |  |
| --- | --- |
| Описание (.h) | Реализация (.cpp) |
| #pragma once#include <SFML/Graphics/RenderWindow.hpp>#include <Map.h>#include <Player.h>#include <Enemy.h>#include <Base.h>#include <Message.h>class Application : private sf::NonCopyable {private: bool gameOver; bool gameStarted; int frags; sf::RenderWindow mWindow; Base mBase; Map map; Player mPlayer; Enemy \*packOfEnemies; Message msgStart; Message msgOver; Message msgLost; Message msgWon;public: Application(); void process\_events(); void update(const sf::Int64 &time); void render();}; | #include <SFML/Window/Event.hpp>#include <Application.h>Application::Application() : mWindow(sf::VideoMode(720, 672), "Battle City"), gameOver(false), gameStarted(false), msgStart(90, 330, "Press \'Enter\' to start"), msgOver(250, 300, "Game over"), msgLost(260, 350, "You lost"), msgWon(265, 350, "You won"), frags(0) { sf::Clock clock; packOfEnemies = new Enemy[4]{ Enemy(52,31), Enemy(147,391), Enemy(532,391), Enemy(628,31) }; while (mWindow.isOpen()) { sf::Int64 time = clock.getElapsedTime().asMicroseconds(); clock.restart(); time /= 800; process\_events(); if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter)) gameStarted = true; if (gameStarted && !gameOver) update(time); render(); }}void Application::process\_events() { sf::Event event; while (mWindow.pollEvent(event)) { switch (event.type) { case sf::Event::Closed: mWindow.close(); break; } }}void Application::update(const sf::Int64 &time) { for (int i(0); i < 4; ++i) if (!packOfEnemies[i].life) ++frags; else { frags = 0; break; } if (frags == 4) gameOver = true; if (!mPlayer.life) gameOver = true; bool collision; for (int i(0); i < 4; ++i) { collision = mPlayer.mSprite.getGlobalBounds().intersects(packOfEnemies[i].mSprite.getGlobalBounds()); if (collision) break; } if (mPlayer.life) mPlayer.update(time, map, collision); for (int i(0); i < 4; ++i) { if (packOfEnemies[i].life) { packOfEnemies[i].update(time, map, collision); if (packOfEnemies[i].bullet.mSprite.getGlobalBounds().intersects(mPlayer.mSprite.getGlobalBounds()) && packOfEnemies[i].bullet.present) { mPlayer.collapse(); packOfEnemies[i].bullet.present = false; } if (packOfEnemies[i].bullet.mSprite.getGlobalBounds().intersects(mBase.mSprite.getGlobalBounds()) && packOfEnemies[i].bullet.present) { mBase.life = false; gameOver = true; } if (mPlayer.bullet.mSprite.getGlobalBounds().intersects(packOfEnemies[i].mSprite.getGlobalBounds()) && mPlayer.bullet.present) { packOfEnemies[i].collapse(); mPlayer.bullet.present = false; } } } if (mPlayer.bullet.mSprite.getGlobalBounds().intersects(mBase.mSprite.getGlobalBounds()) && mPlayer.bullet.present) { mBase.life = false; gameOver = true; }}void Application::render() { mWindow.clear(); map.draw(mWindow); if (mPlayer.life) mWindow.draw(mPlayer.mSprite); if (mPlayer.bullet.present) mWindow.draw(mPlayer.bullet.mSprite); for (int i(0); i < 4; ++i) { if (packOfEnemies[i].bullet.present) mWindow.draw(packOfEnemies[i].bullet.mSprite); if (packOfEnemies[i].life) mWindow.draw(packOfEnemies[i].mSprite); } if (mBase.life) mWindow.draw(mBase.mSprite); if (!gameStarted) msgStart.print(mWindow); if (gameOver) { msgOver.print(mWindow); if (!mBase.life || !mPlayer.life) msgLost.print(mWindow); else msgWon.print(mWindow); } mWindow.display();} |
| #pragma once#include <SFML/System/String.hpp>#include <SFML/Graphics/Texture.hpp>#include <SFML/Graphics/Sprite.hpp>#include <Bullet.h>#include <utility>class Map;class Tank : private sf::NonCopyable {protected: float mX, mY, mWidth, mHeight, mDx, mDy, mSpeed; int mDir; float mCurrentFrame; bool mCollision; sf::String mFile; sf::Texture mTexture;public: explicit Tank(const float &x, const float &y, const float &width, const float &height, const sf::String &file); bool life; Bullet bullet; sf::Sprite mSprite; void animate(const sf::Int64 &time); void collapse(); virtual void move(const sf::Int64 &time) {} void map\_interaction(Map &map); void tank\_interaction();}; | #include <Tank.h>#include <Map.h>Tank::Tank(const float &x, const float &y, const float &width, const float &height, const sf::String &file) : life(true), mX(x), mY(y), mDx(0.f), mDy(0.f), mFile(file), mDir(0), mCollision(false), mCurrentFrame(0.f), mSpeed(0.f), mWidth(width), mHeight(height),  bullet(x, y) { mTexture.loadFromFile(mFile); mSprite.setTexture(mTexture); mSprite.setTextureRect(sf::IntRect(78, 39, mWidth, mHeight)); mSprite.setPosition(x, y);}void Tank::animate(const sf::Int64 &time) { mCurrentFrame += 0.005f \* time; if (mCurrentFrame >= 2) mCurrentFrame -= 2; if (mDx > 0.f) mSprite.setTextureRect(sf::IntRect(0 + 39 \* (int)mCurrentFrame, 0, 39, 39)); if (mDx < 0.f) mSprite.setTextureRect(sf::IntRect(78 + 39 \* (int)mCurrentFrame, 0, 39, 39)); if (mDy > 0.0f) mSprite.setTextureRect(sf::IntRect(0 + 39 \* (int)mCurrentFrame, 39, 39, 39)); if (mDy < 0.0f) mSprite.setTextureRect(sf::IntRect(78 + 39 \* (int)mCurrentFrame, 39, 39, 39));}void Tank::collapse() { life = false; bullet.present = false; mSprite.setPosition(0, 0);}void Tank::map\_interaction(Map &map) { for (int i = mY / 24; i < (mY + mHeight) / 24; ++i) for (int j = mX / 24; j < (mX + mWidth) / 24; ++j) { char tile = map.get\_tile(i, j); if (tile >= '0' && tile <= '2') { if (mDy > 0.f) mY = i \* 24 - mHeight; if (mDy < 0.f) mY = (float)i \* 24 + 24; if (mDx > 0.f) mX = j \* 24 - mWidth; if (mDx < 0.f) mX = (float)j \* 24 + 24; } }}void Tank::tank\_interaction() { if (mDy > 0.f) mY -= 0.3f; if (mDy < 0.f) mY += 0.3f; if (mDx > 0.f) mX -= 0.3f; if (mDx < 0.f) mX += 0.3f;} |
| #pragma once#include <Tank.h>class Enemy : public Tank {private: float timeBeforeMoving; float timeBeforeShot;public: Enemy(const float &x, const float &y); Enemy(const Enemy &copy) : Enemy(copy.mX, copy.mY){} void move(const sf::Int64 &time); void shoot(const float &time); void update(const sf::Int64 &time, Map &map, const bool &collision);}; | #include <Enemy.h>Enemy::Enemy(const float &x, const float &y) : Tank(x, y, 39, 39, "battle-city-src/media/enemySprites.png"), timeBeforeMoving(0.f), timeBeforeShot(0.f) {}void Enemy::move(const sf::Int64 &time) { mSpeed = 0.1f; timeBeforeMoving += time; if (timeBeforeMoving > 1000 + rand() % 501) { mDir = rand() % 4; timeBeforeMoving = 0.f; } switch (mDir) { case 0: mDx = mSpeed; mDy = 0; break; case 1: mDx = -mSpeed; mDy = 0; break; case 2: mDx = 0; mDy = mSpeed; break; case 3: mDx = 0; mDy = -mSpeed; break; } if (mCollision) tank\_interaction(); mX += mDx \* time; mY += mDy \* time;}void Enemy::shoot(const float &time) { timeBeforeShot += time; if (timeBeforeShot > 1000.f) { if (!bullet.present) if (rand() % 2) bullet.present = true; timeBeforeShot = 0.f; }}void Enemy::update(const sf::Int64 &time, Map &map, const bool &collision) { mCollision = collision; move(time); mSpeed = 0.f; mSprite.setPosition(mX, mY); animate(time); map\_interaction(map); bullet.update(map, time, mX, mY, mDir); shoot(time);} |
| #pragma once#include <SFML/Window/Keyboard.hpp>#include <Tank.h>class Player : public Tank {public: Player(); void move(const sf::Int64 &time); void update(const sf::Int64 &time, Map &map, const bool &collision);}; | #include <SFML/Window/Keyboard.hpp>#include <Player.h>#include <Map.h>using sf::Keyboard;Player::Player() : Tank(244, 600, 39, 39, "battle-city-src/media/playerSprites.png") {}void Player::move(const sf::Int64 &time) { if (Keyboard::isKeyPressed(Keyboard::Left)) { mDir = 1; mSpeed = 0.11f; } if (Keyboard::isKeyPressed(Keyboard::Right)) { mDir = 0; mSpeed = 0.11f; } if (Keyboard::isKeyPressed(Keyboard::Up)) { mDir = 3; mSpeed = 0.11f; } if (Keyboard::isKeyPressed(Keyboard::Down)) { mDir = 2; mSpeed = 0.11f; } switch (mDir) { case 0: mDx = mSpeed; mDy = 0; break; case 1: mDx = -mSpeed; mDy = 0; break; case 2: mDx = 0; mDy = mSpeed; break; case 3: mDx = 0; mDy = -mSpeed; break; } if (mCollision) tank\_interaction(); mX += mDx \* time; mY += mDy \* time;}void Player::update(const sf::Int64 &time, Map &map, const bool &collision) { mCollision = collision; move(time); mSpeed = 0.f; mSprite.setPosition(mX, mY); animate(time); map\_interaction(map); bullet.update(map, time, mX, mY, mDir); if (!bullet.present) if (Keyboard::isKeyPressed(Keyboard::Space)) { bullet.timeBeforeShot += time; if (bullet.timeBeforeShot > 10.f) { bullet.present = true; bullet.timeBeforeShot = 0.f; } }} |
| #pragma once#include <SFML/Graphics/Texture.hpp>#include <SFML/Graphics/Sprite.hpp>class Base : private sf::NonCopyable {private: sf::Texture mTexture;public: sf::Sprite mSprite; bool life; Base();}; | #include <Base.h>Base::Base() : life(true) { mTexture.loadFromFile("battle-city-src/media/baseSprite.png"); mSprite.setTexture(mTexture); mSprite.setTextureRect(sf::IntRect(0, 0, 48, 48)); mSprite.setPosition(336, 600);} |
| #pragma once#include <SFML/Graphics/Texture.hpp>#include <SFML/Graphics/Sprite.hpp>class Map;class Bullet {private: float mX, mY, mDx, mDy; int mDir; sf::Texture mTexture;public: bool present; //is it shot? float timeBeforeShot; sf::Sprite mSprite; explicit Bullet(const float &x, const float &y); void shoot() { present = true; } void move(const sf::Int64 &time); void update(Map &map, const sf::Int64 &time, float &x, const float &y, const int &dir); void map\_interaction(Map &map);}; | #include <Bullet.h>#include <Map.h>Bullet::Bullet(const float &x, const float &y) : present(false), mX(x), mY(y), mDir(0), mDx(0.f), mDy(0.f), timeBeforeShot(0.f) { mTexture.loadFromFile("battle-city-src/media/bulletSprite.png"); mSprite.setTexture(mTexture); mSprite.setTextureRect(sf::IntRect(0, 0, 9, 12)); mSprite.setPosition(x, y);}void Bullet::move(const sf::Int64 &time) { switch (mDir) { case 0: mDx = 0.3f; mDy = 0; mSprite.setRotation(90.f); break; case 1: mDx = -0.3f; mDy = 0; mSprite.setRotation(-90.f); break; case 2: mDx = 0; mDy = 0.3f; mSprite.setRotation(180.f); break; case 3: mDx = 0; mDy = -0.3f; mSprite.setRotation(0.f); break; } mX += mDx \* time; mY += mDy \* time; mSprite.setPosition(mX, mY);}void Bullet::update(Map &map, const sf::Int64 &time, float &x, const float &y, const int &dir) { if (present) { move(time); map\_interaction(map); } else { mDir = dir; switch (mDir) { case 0: mX = x + 35.f; mY = y + 15.f; break; case 1: mX = x + 7.f; mY = y + 23.f; break; case 2: mX = x + 23.f; mY = y + 32.f; break; case 3: mX = x + 15.f; mY = y + 5.f; break; } mSprite.setPosition(mX, mY); }}void Bullet::map\_interaction(Map &map) { for (int i = mY / 24; i < (mY + 12) / 24; ++i) for (int j = mX / 24; j < (mX + 9) / 24; ++j) { char tile = map.get\_tile(i, j); if (tile == '0' || tile == '2') present = false; if (tile == '1') { map.break\_wall(i, j); present = false; } }} |
| #pragma once#include <SFML/System/String.hpp>#include <SFML/Graphics/Texture.hpp>#include <SFML/Graphics/Sprite.hpp>#include <SFML/Graphics/RenderWindow.hpp>class Map : private sf::NonCopyable {private: static const int HEIGHT\_MAP = 28; static const int WIDTH\_MAP = 30; sf::String \*TileMap; sf::Texture mTexture; sf::Sprite mSprite;public: Map(); void break\_wall(const int &i, const int &j); const char get\_tile(const int &i, const int &j) { return TileMap[i][j]; } void draw(sf::RenderWindow &window);}; | #include <Map.h>Map::Map() { TileMap = new sf::String[HEIGHT\_MAP] { "000000000000000000000000000000", "00 00", "00 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 112211 11 11 00", "00 11 11 112211 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 00", "00 11 11 11 11 00", "00 11 11 00", "00 11 11 00", "0011 1111 1111 1100", "0022 1111 1111 2200", "00 11 11 00", "00 111111 00", "00 11 11 111111 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 11 11 00", "00 11 11 11 11 00", "00 11 11 11 11 00", "00 11 11 1111 11 11 00", "00 1 1 00", "00 1 1 00", "000000000000000000000000000000", }; mTexture.loadFromFile("battle-city-src/media/groundSprites.png"); mTexture.setSmooth(true); mSprite.setTexture(mTexture);}void Map::break\_wall(const int &i, const int &j) { TileMap[i][j] = ' ';}void Map::draw(sf::RenderWindow &window) { for (int i(0); i < HEIGHT\_MAP; i++) for (int j(0); j < WIDTH\_MAP; j++) { switch (TileMap[i][j]) { case ' ': mSprite.setTextureRect(sf::IntRect(0, 0, 24, 24)); break; case '0': mSprite.setTextureRect(sf::IntRect(72, 0, 24, 24)); break; case '1': mSprite.setTextureRect(sf::IntRect(24, 0, 24, 24)); break; case '2': mSprite.setTextureRect(sf::IntRect(48, 0, 24, 24)); break; } mSprite.setPosition(j \* 24, i \* 24); window.draw(mSprite); }} |
| #pragma once#include <SFML/Graphics/Text.hpp>#include <SFML/Graphics/RenderWindow.hpp>class Message {private: float mX, mY; sf::Font mFont; sf::Text mStr;public: explicit Message(const float &x, const float &y, const sf::String &str); void print(sf::RenderWindow &window) { window.draw(mStr); }}; | #include <Message.h>Message::Message(const float &x, const float &y, const sf::String &str) : mX(x), mY(y) {  mFont.loadFromFile("battle-city-src/media/PressStart2P.ttf"); mStr.setFont(mFont); mStr.setString(str); mStr.setPosition(mX, mY); mStr.setCharacterSize(25);} |

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы я закрепил навыки применения ООП на языке C++ и важные части мультимедийной библиотеки SFML, необходимые для работоспособности программ, разработанных на её основе, а также научился управлять собственными проектами в Системе Контроля Версий Git.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Arthur Moreira. SFML Game Development [Текст]/ Jan Haller, Henrik Vogelius Hansson. 2013. – 296 с.
2. Difference Between Git and GitHub – Eduo Shaun // Medium. [2017-2019]. Дата обновления: 9.02.2017. URL: [https://medium.com/@eduoshaun/difference-between-git-and-github-807f1a57d438](https://medium.com/%40eduoshaun/difference-between-git-and-github-807f1a57d438) (дата обращения: 16.06.2019).
3. Интегрированная среда разработки // Википедия. [2019—2019]. Дата обновления: 25.05.2019. URL: <https://ru.wikipedia.org/?oldid=100020793> (дата обращения: 16.06.2019).
4. Уроки по sfml, форум, вики, чат, командные проекты // kychka-pc. [2015-2019]. Дата обновления: 25.07.2019. URL: <http://kychka-pc.ru/> (дата обращения: 16.06.2019).

ПРИЛОЖЕНИЕ А. ДИАГРАММА КЛАССОВ

ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД ТЕСТОВОГО ПРИЛОЖЕНИЯ

Код программы доступен по ссылке: <https://github.com/xt1zer/battle-city/tree/master/>.