**

Колледж космического машиностроения и технологии

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка игры "Танчики"»

Выполнил студент

Стрельников Сергей Дмитриевич

Группа П1-17

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королёв 2020 г.

Оглавление

[Введение 4](#_Toc43936356)

[1. Теоретическая часть 5](#_Toc43936357)

[1.1. Описание предметной области 5](#_Toc43936358)

[1.1.1. Составляющие персонажа и противников («Танчики») 7](#_Toc43936359)

[1.2. Описание существующих разработок 11](#_Toc43936360)

[2. Проектная часть 13](#_Toc43936361)

[2.1. Диаграмма прецедентов 13](#_Toc43936362)

[2.2. Выбор инструментов 15](#_Toc43936363)

[2.2.1. Библиотека SFML 16](#_Toc43936364)

[2.2.2. Текстовый редактор Sublime Text 3 16](#_Toc43936365)

[2.2.3. Компилятор GNU GCC 16](#_Toc43936366)

[2.3. Проектирование сценария 16](#_Toc43936367)

[2.4. Диаграмма классов 19](#_Toc43936368)

[2.5. Описание главного модуля 22](#_Toc43936369)

[2.6. Описания спецификаций к модулям 23](#_Toc43936370)

[2.7. Описание модулей 25](#_Toc43936371)

[2.8. Описание тестовых наборов модулей 34](#_Toc43936372)

[2.9. Описание применения средств отладки 37](#_Toc43936373)

[2.10. Анализ оптимальности использования памяти и быстродействия 39](#_Toc43936374)

[3. Эксплуатационная часть 40](#_Toc43936375)

[3.1. Руководство оператора 40](#_Toc43936376)

[3.1.1. Назначение программы 40](#_Toc43936377)

[3.1.2. Условия выполнения программы 40](#_Toc43936378)

[3.1.3. Выполнение программы 41](#_Toc43936379)

[3.2. Выполнение программы 44](#_Toc43936380)

[3.2.1. Выполнение функции начала новой игры 44](#_Toc43936381)

[3.2.2. Функция выбора «Танчика» 45](#_Toc43936382)

[3.2.3. Функция игрового процесса 45](#_Toc43936383)

[3.3. Завершение программы 46](#_Toc43936384)

[Заключение 47](#_Toc43936385)

[Список литературы и интернет-источников 48](#_Toc43936386)

[Приложение 1. Модуль Game.cpp. 49](#_Toc43936387)

[Приложение 2. Модуль MainMenu.hpp. 49](#_Toc43936388)

[Приложение 3. Модуль MainMenu.cpp. 49](#_Toc43936389)

[Приложение 4. Модуль mainGame.hpp. 52](#_Toc43936390)

[Приложение 5. Модуль mainGame.cpp. 52](#_Toc43936391)

[Приложение 6. Модуль ClassTanks.cpp. 58](#_Toc43936392)

[Приложение 7. Модуль ClassIFV.cpp. 63](#_Toc43936393)

[Приложение 8. Модуль ClassMBT.cpp. 67](#_Toc43936394)

[Приложение 9. Модуль ClassBMPT.cpp. 68](#_Toc43936395)

[Приложение 10. Модуль ClassOBJ.cpp. 72](#_Toc43936396)

[Приложение 11. Модуль ButtonsClass.cpp. 72](#_Toc43936397)

[Приложение 12. Модуль Map.cpp. 73](#_Toc43936398)

Введение

Целью данного курсового проекта является написание 2D аркадной видеоигры «Танчики». В современном мире создание видеоигр является одним из наиболее крупных сегментов индустрии развлечений. С каждым годом игры показывают, что могут являться способом творческого самовыражения, не уступающим литературе или живописи, а финансовый успех таких игровых франшиз, как Call of Duty или Assassin’s Creed – живое доказательство тому, что индустрия игр развивается очень быстро.

Все больше людей придерживаются мнения, что игры – это не просто искусство, но и вообще его абсолют, объединяющий в себе остальные проявления творчества.

Выбранная темя является желанием понимать основные принципы создания игр и возможность самому написать интересный продукт с оглядкой на существующие решения.

В первой части будет рассмотрена предметная область данной темы, а также несколько уже существующих разработок.

Во второй части будут рассмотрены диаграммы прецедентов и классов, выбранные инструменты и разработанные модули, структура программы и листинг главных частей модулей.

В третьей части будет рассмотрено руководство пользователя.

В заключительной части будут сделаны выводы о проекте.

1. Теоретическая часть
   1. Описание предметной области

Аркада [1] — жанр компьютерных игр, характеризующийся коротким по времени, но интенсивным игровым процессом.У жанра “аркад” есть множество поджанров, включающих в себя платформеры, простейшие симуляторы, файтинги, и шутеры

Классические аркады характеризуются следующими свойствами:

* ***игра на одном экране*.** Все действия происходят на одном экране чтобы игроки в любой момент времени могли видеть весь игровой мир и принимать решения. У многих игр было больше одного экрана, каждый экран был отдельным уровнем**.** Примеры таких игр: *Joust*, *Pac-Man*, *Mario Bros.*
* ***бесконечная игра.*** Заключается в том, что игроки могут играть в аркаду бесконечное время, и соответственно, не могут выиграть. Игроков удерживало то, что они делали вызов сами себе – на сколько долго можно продержаться. Относительно геймдизайна в аркадах игрок никогда не выигрывал, и каждая игра заканчивалась поражением. Со временем играть становилось все сложнее из-за повышающего уровня слолжности
* ***множество жизней***. Обычно, классическая аркада предлагает игроку несколько попыток. Это делается для того, чтобы игрок анализировал свои ошибки, учился на них и не проходил игру или уровень заново.
* ***игровой счёт.*** Практически все классические аркады включают в себя игровой счёт. Когда игрок получает очки за выполнение различных целей
* ***быстрое обучение****, простой игровой процесс*. Для многих аркад характерен не сложный игровой процесс, в который легко понять даже новичку. От профессионала его будет отличать лишь умение грамотно пользоваться небольшим количеством функций
* ***нет сюжета/истории***. Во многих, но не во всех аркадах, нет сюжета. Он попросту не нужен, потому что игрок сам в состоянии понять, что происходит на экране, и сам игровой процесс не нуждается в какой-либо истории.
* ***искусственный интеллект*** (ИИ) аркадных игр как правило, представляет собой простые действия и не реагирует на поведение игрока. И, зачастую, сложность увеличивается просто количеством противников
  + 1. Составляющие персонажа и противников («Танчики»)

Все представленные в курсовом проекте текстуры «Танчиков» срисованы с реальных образцов военной техники

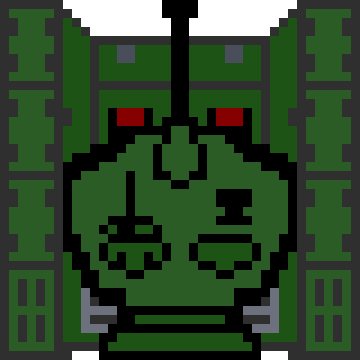
. 

Рисунок 1. «Танчик Т-90»

«Танчик Т-90» (Рис.1) срисован с российского основного боевого танка Т-90 «Владимир» (Рис.2)



Рисунок 2. ОБТ Т-90 «Владимир»

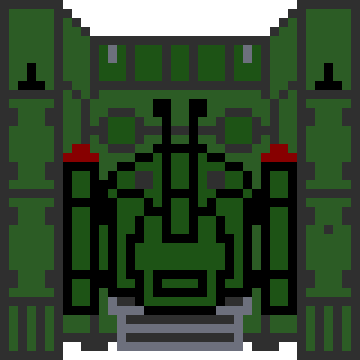


Рисунок 3. «Танчик БМПТ «Терминатор»»

**«**Танчик БМПТ «Терминатор»» (Рис.3) срисован с российской боевой

машины поддержки танков БМПТ-2 «Терминатор» (Рис.4)



Рисунок 4. БМПТ-2 «Терминатор»

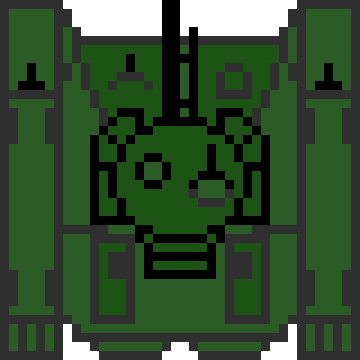


Рисунок 5. «Танчик БМП-3»

«Танчик БМП-3» (Рис.5) срисован с российской боевой машины пехоты БМП-3(Рис.6)



Рисунок 6. БМП-3

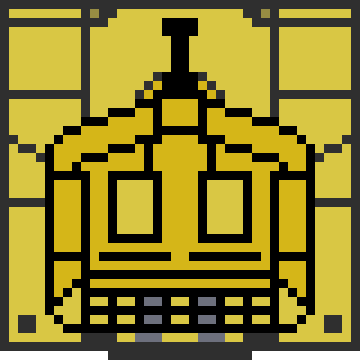


Рисунок 7. «Танчик М1 «Абрамс»»

«Танчик М1 «Абрамс»» (Рис.7) срисован с основного боевоготанкаСША M1A2 «Abrams» (Рис.8)



Рисунок 8. M1A2 «Abrams»

* 1. Описание существующих разработок

В основе курсового проекта лежат механики из уже существующих

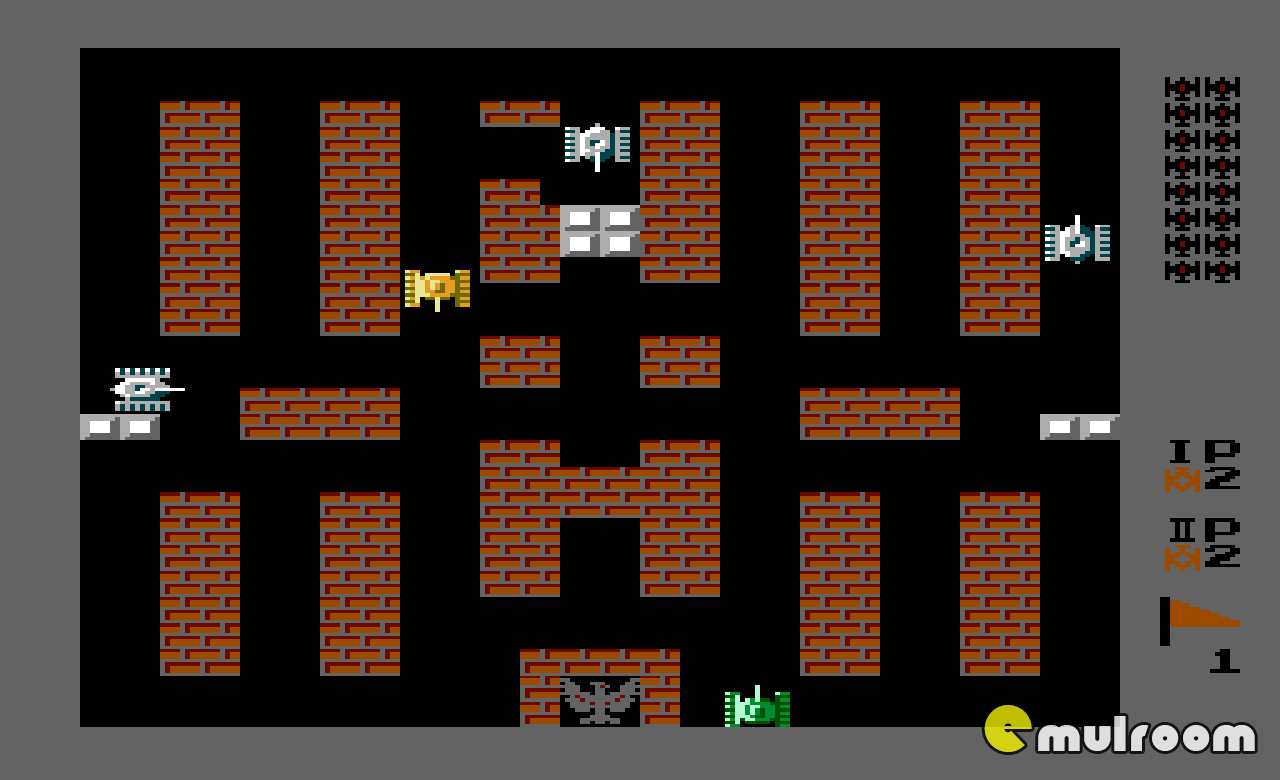
проектов.

В их числе аркада 1980 года Battle City[2] (Рис.9)

Battle City – это аркадная игра в которой основной задачей является защита базы от танков противника. В процессе игры на поле будут появляться различные бонусы: гранаты, лопаты, дополнительные жизни и другое. К преимуществам данной игры можно отнести достаточно затягивающий игровой процесс, возможность редактировать карты и приятную графику несмотря на то, что она устарела. На мой взгяд недостатки отсутствую. Игра является классической и все недостатки пришли только из-за технического устаревания игры

Из игры Battle City в курсовом проекте присутствуют следующие механики:

* Полигон действий виден сверху
* Управлением игроком «Танчиком»
* Появление врагов в разных местах карты



* + - Рисунок 9. Battle City

Аркадный танковый экшн Armored Warfare 2015[3] года (Рис.10).

Представляет собой попытку создать конкурента игре World of Tanks только в современном сеттинге. Разрабатывалась obsidian entertainment, после была передана mail.ru group. Игровой процесс это – онлайн бой 15 на 15 игроков на разных видах современной бронетехнике. К преимуществам можно отнести достаточно динамичный игровой процесс, большое разнообразие бронетехники: от тяжелых основных боевых танком, до легких машин разведки.

За достаточно красивую графику отвечает движок Cry Engine 3. Но у такого решения есть недостаток – это плохая работоспособность на слабых ПК. Так же к недостаткам можно отнести навязывание финансовых влияний для комфортной игры: увеличение дохода, ускоренное развитие, покупка уникальной, не прокачиваемой техники. Этот факт достаточно сильно бьет по престижу в игровом сообществе, что привело к маленькой аудитории игр



Рисунок 10. Вид из ангара в Armored Warfare

Из игры Armored Warfare были взяты следующие решения:

* Механика ПТУРов (Управляемых снарядов)
* Механика Авто-Пушек
* Механика неуправляемых снарядов

1. Проектная часть
   1. Диаграмма прецедентов

Для определения вариантов использования к проекту была построены диаграммы прецедентов (Рис.11).

Диаграмма прецедентов «Пользователь» показывает возможные действия пользователя после запуска приложения. Пользователь может запустить саму игру, просмотреть окно помощи или выйти из приложения



Рисунок 11. Диаграмма прецедентов Пользователь

Диаграмма прецедентов «Игрок» (Рис.12) показывает возможные действия игрока во время игры. Игрок может выбрать «Танчик» и в последствии управлять им, уничтожать противников из доступных вариантов вооружения и получать за это очки.

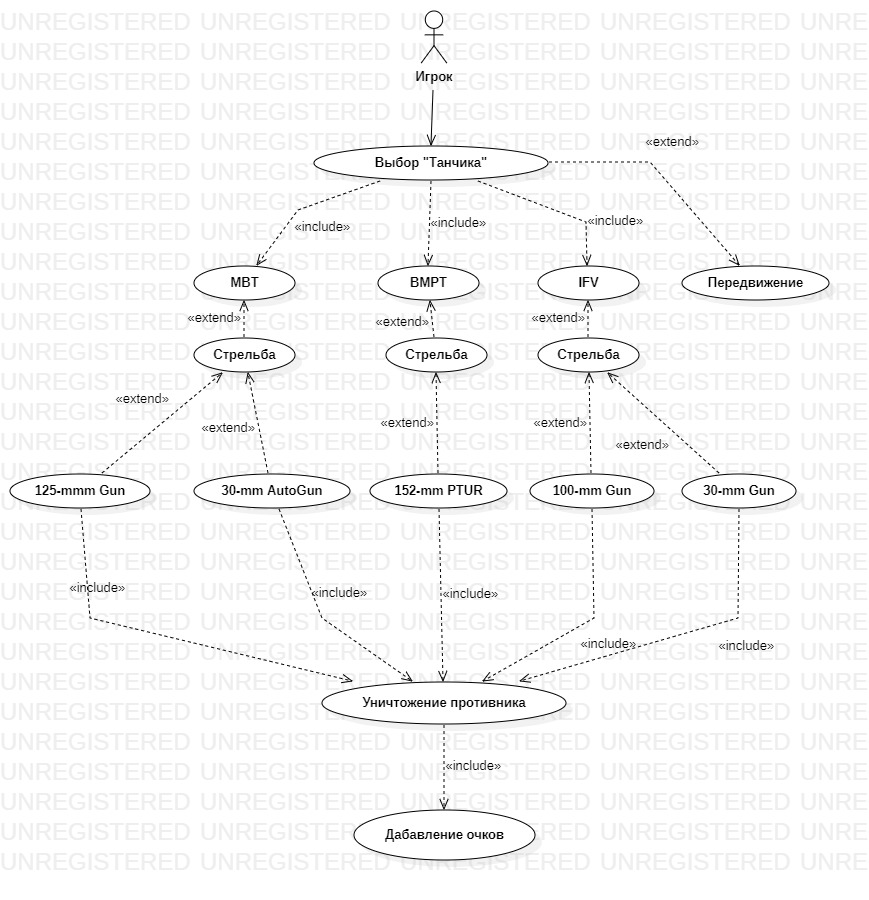


Рисунок 12. Диаграмма прецедентов Игрок

* 1. Выбор инструментов

Для написания проекта были выбраны: библиотека SFML[4], текстовый редактор Sublime Text 3[5] и компилятор языка C++ GNU GCC[6] на основе следующих таблиц сравнения

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструментов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Участие в корпоративном  проекте | Простота  сопровождения | Наличие библиотек | Наличие  Документации  На русском | Скорость разработки |
|  | Высокая | Средняя | Высокая | Высокая | Средняя |

Таблица 2. Критерии выбора инструментов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Простота синтаксиса | Популярность | Простота  использования | Интерфейс |
|  | Высокая | Низкая | Ниже средней | Низкая |

Таблица 3. Оценка библиотек

|  |  |  |  |
| --- | --- | --- | --- |
|  | SFML | SDL | GTK+ |
| Простота синтаксиса | 7 | 3 | 7 |
| Документация на Русском языке | 10 | 6 | 8 |
| Итог: | 17 | 9 | 5 |

Таблица 4. Оценка сред разработки

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sublime Text 3 | Atom | Vim |
| Интерфейс | 9 | 9 | 6 |
| Простота использования | 6 | 3 | 3 |
| Популярность | 6 | 4 | 10 |
| Итог: | 21 | 16 | 19 |

Таблица 5. Оценка языков программирования

|  |  |  |  |
| --- | --- | --- | --- |
|  | Python | C++ |  |
| Участие в корпоративном проекте | 8 | 10 |  | |
| Простота сопровождения | 9 | 5 |  | |
| Наличие библиотек | 10 | 7 |  | |
| Скорость разработки | 7 | 9 |  | |
| Документация на Русском языке | 4 | 8 |  | |
| Итог: | 28 | 29 |  | |

* + 1. Библиотека SFML

SFML это графическая библиотека для создания простой 2-d

графики и спрайтовой анимации. Для написания проекта использовались модули:

* sfml-graphics
* sfml-system
* sfml-window
  + 1. Текстовый редактор Sublime Text 3

Sublime text 3 – это проприетарный текстовый редактор.

Функциональность придают такие возможности как быстрая навигация, подсветка синтаксиса, высокая степень настраиваемости, богатые визуальные схемы.

* + 1. Компилятор GNU GCC

Свободно распространяемый компилятор языка с++. Позволяет получать полную и точную информацию о работе о программе при отладке.

* 1. Проектирование сценария

Данная программа ориентирована на следующие 2 последовательных сценария использования:

1. Сценарий пользователя при запуске приложения (Рис.13):

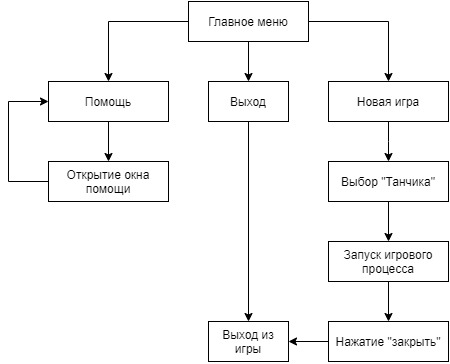


Рисунок 13. Сценарий использования

После запуска программы пользователь имеет следующие варианты действия

* Запуск новой игры
* Просмотр окна помощи
* Выход из программы

1. Сценарий пользователя при запуске игрового процесса (Рис.14):

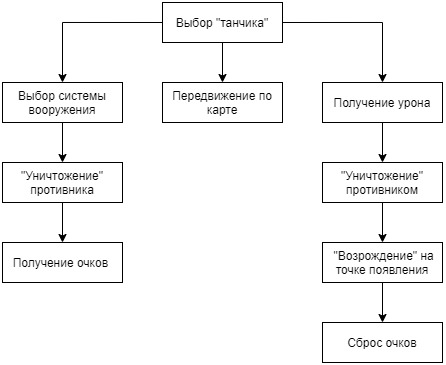


Рисунок 14. Сценарий использования

После запуска игрового процесса и выбора «Танчика» игрок имеет

Следующие варианты действия:

* Передвижение по игровой карте
* Выбор системы вооружения
* «Уничтожение» противников
* Получение урона от противников
* «Уничтожение» противником

После «Уничтожения» противника игрок получает фиксированное количество очков к своему счёту.

После «Уничтожения» противником игрок перемещается на место старта и происходит сброс набранных очков

* 1. Диаграмма классов

После определения функциональных требований к проекту, была описана предметная область с целью построения диаграммы классов

(Рис.15).

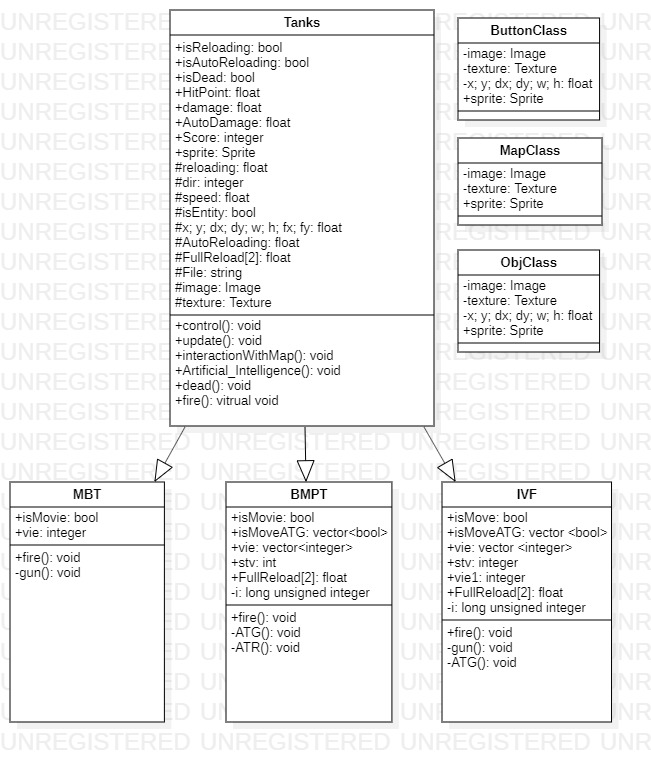


Рисунок 15 Диаграмма классов

**Класс «Таnks»** содержит**:**

публичные поля isReload, isAutoReload, isDead, HitPoint, damage, AutoDamage, Score, sprite;

защищенные поля reloading, dir, speed, isEntity, x, y, dx, dy, w, h, fx, fy, AutoReloading, FullReload[2], File, images, texture;

публичные методы:

* control(), которые отвечает за управление «Танчиком»;
* update(), который отвечает за перемещение «Танчика»;
* interactionWithMap(), который отвечает за столконовение с препятствиями;
* Artificial\_Intelligence(), который отвечает за передвижение противников;
* dead(), который отвечает за уничтожения танчика, является перегруженным;
* fire(), который отвечает за методы стрельбы, является виртуальным;

**Класс «MBT****»** является наследованным от класса «Tanks» и содержит:

публичные поля isMovie, vie;

приватные метод:

* gun(), который отвечает за стрельбу неуправляемым снарядом

публичный метод

* fire(), который отвечает за вызов приватного метода стрельбы

**Класс «BMPT»** является наследованным от класса «Tanks» и содержит:

публичные поля isMovie, isMovieATG, vie, stv, FullReload[2];

приватные поля I;

приватные методы:

* ATG(), который отвечает за стрельбу очередями
* ATR(), который отвечает за стрельбу управляемыми снарядами

Публичные методы:

* fire(), который отвечает за вызов приватных методов

**Класс «IFV»** является наследованным от класса «Tanks» и содержит:

публичные поля isMovie, isMovieATG, vie, vie2, stv, FullReload[2];

приватные поля I;

приватные методы:

* ATG(), который отвечает за стрельбу очередями
* gun(), который отвечает за стрельбу неуправляемыми снарядами

Публичные методы:

* fire(), который отвечает за вызов приватных методов

**Класс «ButtonClass»** содержит:

приватные поля image, texture, x, y, dx, dy, w, h;

публичные поля sprite;

**Класс «MapClass»** содержит:

приватные поля image, texture;

публичные поля sprite;

**Класс «ObjClass»** содержит:

приватные поля image, texture, x, y, dx, dy, w, h;

публичные поля sprite;

* 1. Описание главного модуля

Главный модуль представляет с собой файл Game.cpp.

В главный модуль разработки входит вызов функции главного меню, которая отвечает за дальнейшее использование приложения (Рис.16).

Данный модуль запускается единожды и является самым первым в запуске приложения. Состоит из кода, который приведён в листинге 1.

**Листинг 1:**

#include <iostream>

#include <vector>

#include <cstring>

#include <sstream>

#include <SFML/Graphics.hpp>

//Подключение файла содержащий функции главного меню

#include "resourse/MainMenu.hpp"

using namespace std;

using namespace sf;

int main()

{

//Вызов функции главного меню

Launcher();

return 0;

}

**Блок-схема к главному модулю:**

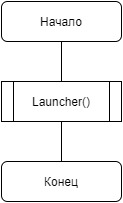


Рисунок 16. Блок-схема

* 1. Описания спецификаций к модулям

В данном пункте описаны все публичные методы модулей курсового проекта.

Описание всех публичных членов модулей курсового проекта

**Главный модуль Game.cpp**

В главном модуле только вызывается функция главного меню.

**Модуль главного меню MainMenu.cpp**

mass – вектор всех объектов изображений и кнопок главного меню

window – объект окна меню

CurretFrame – переменная, отвечающая за смену кадров

Press() – отвечает за подсветку кнопок;

FrameMenu() – отвечает за смену кадров анимации фона;

Launcher() – отвечает за работу главного меню;

…

**Модуль игрового процесса mainGame.cpp**

window – объект окна;

clock – объект внутриигрового времени;

Scr – поле, отвечающее за отображение игровых данных в окне;

player - объект игрока;

playerShell – объект снаряда игрока. Может быть управляемым и неуправляемым;

playerBMPT – вектор объектов снарядом кассеты игрока ;

entity – вектор объектов противников;

entityShell – вектор объектов снарядов противников;

Press() – отвечает за подсветку при выборе «Танчика»;

MapFunct() – отвечает за отрисовку карты;

Choice() – отвечает за выбор «Танчика»;

isFire() – отвечает за вывод статуса перезарядки;

Score() – отвечает за подчёт очков и их вывода;

Game() – отвечает за запуск игрового процесса ;

…

**Модуль родительского класса Tanks.hpp**

damage – поле урона управляемым или неуправляемым снарядом;

AutoDamage – поле урона снарядом авто пушки;

HitPoint – поле кол-ва возможного полученного урона;

Score – поле полученных очков;

control() – отвечает за управление «Танчиком»;

update() – отвечает за передвижение «Танчиков»;

interactionWithMap() - отвечает за проверку столкновения с препятствиями;

Artificial\_Intelligence() – отвечает за передвижение противников;

dead() – отвечает за обработку попаданий по «Танчикам». Является перегруженной;

fire() – отвечает за функции ведения огня «Танчиков». Является виртуальной;

**Модуль дочернего класса ClassMBT.hpp**

vie – поле, отвечающее за направление полёта неуправляемого снаряда;

Fire() – отвечает за функцию ведения огня «Танчика» класса MBT;

…

**Модуль дочернего класса ClassBMPT.hpp**  
 isMovieATG – вектор, отвечающий за проверку движения у каждого снаряда в очереди

vie – вектор, отвечающий за направление движения каждого снаряда в очереди

Fire() – отвечает за функцию ведения огня «Танчика» класса BMPT

…

**Модуль дочернего класса ClassIFV.hpp**

Fire() – отвечает за функцию ведения огня «Танчика» класса IFV

isMovieATG – вектор, отвечающий за проверку движения у каждого снаряда в очереди

vie – вектор, отвечающий за направление движения каждого снаряда в очереди

…

**Модуль карты map.hpp**

HEIGHT\_MAP – поле высота карты

WIDHT\_MAP – поле ширина карты

TileMap – поле карты

…

* 1. Описание модулей

В данном пункте будут описаны все оставшиеся модули курсового проекта.

* Модуль главного меню MainMenu.cpp

Данный модуль представляет собой набор функции для создания и отрисовки заднего фона, анимации, кнопок, а также вызов определенных действий для каждой кнопки, в том числе вызов модуля игрового процесса (Рис.17).

**Листинг 2. Функции MainMenu.cpp**

//Подсветка кнопок

//Приниается вектор всех кнопок и вектор координат курсора

void press(vector<Button\*> v, Vector2f pos)

//Смена кадров фона

//Принимется фон окна, время, номер кадра

void FrameMenu(Button\* menu, float time, float &CurrentFrame)

//Основная фунция главного меню

void launcher()

**Листинг 3. Функция press()**

//Подсветка кнопок

//Приниается вектор всех кнопок и вектор координат курсора

void press(vector<Button\*> v, Vector2f pos)

{

//i = 1, кроме 1 элемента, это фон

for(long unsigned int i = 1; i < v.size(); i++)

{

//Если мышка неведена на кнопку, то она подсвечивается черным

//(!= 5) кроме 5 элемента, это окно помощи

if(v[i]->sprite.getGlobalBounds().contains(pos.x, pos.y) && i != 5)

v[i]->sprite.setColor(Color::Black);

else if (i != 5)

v[i]->sprite.setColor(Color::White);

}

}

**Блок-схема к функции подсветки кнопок:**

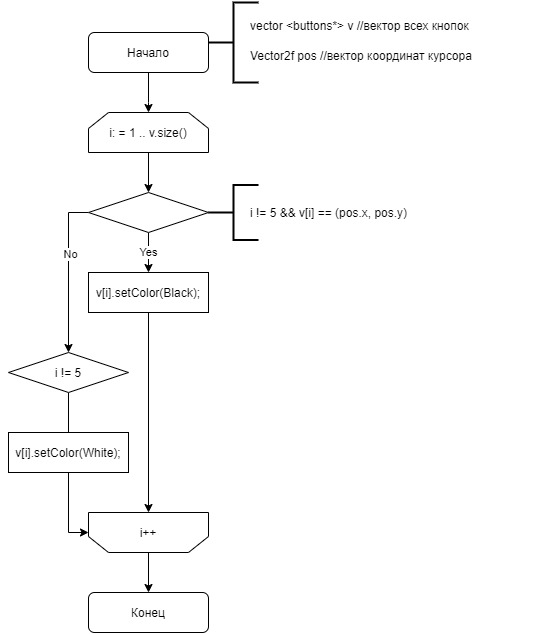
****

Рисунок 17. Блок – схема функции press

**Листинг 4. Функция FrameMenu()**

//Смена кадров фона

//Принимается фон окна, время, номер кадра

void FrameMenu(Button\* menu, float time, float &CurrentFrame)

{

CurrentFrame += 0.005 \* time;

if(CurrentFrame > 3)

CurrentFrame = 0;

menu->sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) \* 1280), 0, 1280, 720));

}

**Листинг 5. Функция launcher()**

//Основная фунция главного меню

void launcher()

{

RenderWindow window(VideoMode(1280, 720), "Launcher"); //создание окна 1280х720

float CurrentFrame = 0; //Номер кадра

bool isOpen = true; //открыто ли окно помощи

Clock clock;

vector<Button\*> mass; //вектор кнопок

mass.push\_back(new Button("Main\_menu.png", 0, 0, 1280, 720, 0, 0)); //Фон [0]

mass.push\_back(new Button("Buttons.png", 2, 2, 200, 50, 1000, 80)); //Новая игра [1]

mass.push\_back(new Button("Buttons.png", 2, 52, 200, 50, 1000, 130)); //Редактор [2]

mass.push\_back(new Button("Buttons.png", 2, 102, 200, 50, 1000, 180)); //Помощь [3]

mass.push\_back(new Button("Buttons.png", 2, 152, 200, 50, 1000, 230)); //Выход [4]

mass.push\_back(new Button("Buttons.png", 202, 2, 240, 280, 750, 80)); //Окно помощи[5]

mass[5]->sprite.setColor(Color(0, 0, 0, 1));

//Покраска окна помощи в прозрачный

// Ниже самое важное, малоизменяемое

while(window.isOpen())

{

Vector2i pixelPos = Mouse::getPosition(window);

//забираем коорд курсора

Vector2f pos = window.mapPixelToCoords(pixelPos);

//переводим их в игровые (уходим от коорд окна)

srand(time(NULL));

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

clock.restart();

time = time / 800;

Event close;

while(window.pollEvent(close))

{

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

window.close();

// Тут выбор пункта меню

else if(Mouse::isButtonPressed(Mouse::Left))

{

if(mass[1]->sprite.getGlobalBounds().contains(pos.x, pos.y)) //Если нажата "Новая игра" закрываем окно лайнчера и запускаем игру

{

window.close();

game();

}

if(mass[3]->sprite.getGlobalBounds().contains(pos.x, pos.y))

//Если нажата "Помощь" то, окно помощи становится видимым

{

if(isOpen == false)

{

mass[5]->sprite.setColor(Color(0, 0, 0, 1)); //при повторном нажатии окно становится невидимым

isOpen = true;

}

else

{

mass[5]->sprite.setColor(Color(255, 255, 255, 255));

isOpen = false;

}

}

if(mass[4]->sprite.getGlobalBounds().contains(pos.x, pos.y)) //При нажатии выход, лаунчер закрывется и ничего не происходи

{

window.close();

}

}

}

/// Выше самое важное, малоизменяемое

press(mass, pos);

FrameMenu(mass[0], time, CurrentFrame);

// Рисуем всё, что есть

window.draw(mass[0]->sprite);

for(long unsigned int i = 1; i < mass.size(); i++)

{

window.draw(mass[i]->sprite);

}

window.display();

}

}

* Модуль игрового процесса mainGame.cpp

В модуле описаны функции выбора «Танчика», отрисовки окна выбора «Танчика», отрисовки карты. Функция создание полей для записи и вывода на экран очков, рекорда очков, остатка попыток и вызова отдельной функции для расчета статуса перезаряди.

Имеется главная функция игрового процесса, в которой создаются объекты игрока, противников и их снарядов. Вызываются методы управления «Танчиками», рисовки карты и всех объектов, создания и удаления новых противников.

**Листинг 6. Функции mainGame.cpp**

//Подсветка кнопок

//Приниается вектор всех кнопок и вектор координат курсора

void press(vector<obj\*> v, Vector2f pos)

//Отрисовка карты

//Принимается объект карты, граница по х, граница по у

void MapFunct(MapClass \*MapObjF, int i , int j)

//Выбор Танчика

//Принимается переменная выбора

void choice(int &a)

//Фунция готовности к стрельбе

//Принимается переменная перезарядки

string isFire(bool isReloadingF)

//Фунция записи очков, максимума очков, перезарядки

//Принимается объект текста, очки, максимальные очки, кол-во жизней, //переменная перезарядки 1 и 2 вида оружия

void Score(Text \*text, int ScoreF, int \*MaxScoreF, float HitPointF, bool isReloadingF, bool isAutoReloadingF)

//Основная функция игры

void game()

* Модуль родительского класса ClassTanks.cpp

Представляет собой главный класс, в котором описаны публичные методы управления, передвижения, проверки столкновения с препятствиями, поведения противников. Перегруженные методы проверки уничтожения и виртуальный метод стрельбы. Является родительским классом для остальных классов «Танчиков».

**Листинг 7. Методы ClassTanks.hpp**

//Метод управления

//Принимает время

void control(float time)

//Метод передвижения

//Принимаеися время

void update(float time)

//Метод проверки столкновения

void interactionWithMap()

//Метод противников

//Принимается время

void Artificial\_Intelligence(float time)

//Перегруженный метод засчитывания попадания

//Принимается объкт снаряда, переменные места возраждения, получаемый урон, вектор объектов автопушки, получаемый урон Автопушки,

//Объект попадания обычного снаряд и объект попадания снаряда автопушки

//Толко для противников

void dead(obj\* Shell, float a, float b, float Fdamage, vector<obj\*> AutoGun, float FdamageA, obj\* fireAutoGun, obj\* fireGun)

//Перегруженный метод засчитывания попадания

//Принимается объкт снаряда, переменные места возраждения, получаемый урон, объект попадания обычного снаряда

//Аналогично первому методу

// только для игрока

void dead(obj\* Shell, float a, float b, float Fdamage, obj\* fireGun)

//Виртульная фунция для стрельбы

virtual void fire(obj\* shell, float time, vector<obj\*> AutoGun)

* Модуль дочернего класса MBT ClassMBT.cpp

В данном классе описан публичный метод стрельбы, в котором вызывается приватный метод. Данный метод является реализацией неуправляемой стрельбы. Класс является наследником главного класса.

**Листинг 8. Методы ClassMBT.hpp**

//Метод неуправляемого

//Выпускается один снаряд без возможноти управления

//Принимает объект снаряда и время

void gun(obj\* shell, float time)

//Метод стрельбы

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

* Модуль дочернего класса BMPT ClassBMPT.cpp

В этом классе так же описан публичный метод стрельбы, в которым вызывается два приватных метода. Первый метод представляет собой реализацию стрельбы управляемым снарядом. Второй метод являет реализацией стрельбы кассетами по 5 снарядов. Класс является наследником главного класса

**Листинг 9. Методы ClassBMPT.hpp**

//Метод Противотанковой Управляемой Ракеты(ПТУР)

//Принимает ПТУР и время

//Всего 4 ракеты, которые вылетают поочереди и могут управляться игроком

//Аналогичен неуправляемой стрельбе

void ATR(obj\* shell, float time)

//Метод Автоматической Стрельбы

//В касете находится 5 снарядов с небольшой перезарядкой между залпами.

// После отсртрела 5 снарядов начинается долгая перезарядка всей касеты

// Принимает вектор снарядом и время

// Алгоритм аналогичен стрельбое неуправляемым снарядом, только для 5-ти элементов

void ATG(vector<obj\*> shell, float time)

//Метод стрельбы

//Заполняет вектора вида и проверки движения для АвтоПушки

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

* Модуль дочернего класса IFV ClassIFV.cpp

Этот класс имеет аналогичный публичный метод стрельбы, в котором так же вызываются два приватных метода. Первый метод аналогичен реализацией неуправляемому снаряду. Второй аналогичен реализацией стрельбы кассетами по 5 снарядов. Класс является наследником главного класса.

**Листинг 10. Методы ClassIFV.hpp**

//Метод Автоматической Стрельбы

//В касете находится 5 снарядов с небольшой перезарядкой между залпами.

// После отсртрела 5 снарядов начинается долгая перезарядка всей касеты

// Принимает вектор снарядом и время

// Алгоритм аналогичен стрельбое неуправляемым снарядом, только для 5-ти элементов

void ATG(vector<obj\*> shell, float time)

//Метод неуправляемого снаряда

//Выпускается один снаряд без возможноти управления

//Принимает объект снаряда и время

void gun(obj\* shell, float time)

//Метод стрельбы

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

* Модуль карты map.hpp

Представляет собой класс для создания спрайтов карты из указанного файла с установкой размера одного title карты без установки начально местоположения

* Модуль класса для кнопок ButtonsClass.hpp

Представляет собой класс для создания спрайтов кнопок из указанного файла, задание местоположения.

* Модуль класса для объектов ClassObj.hpp

Представляет собой класс для создания спрайтов объектов из указанного файла, установки их границ и задание местоположения.

* 1. Описание тестовых наборов модулей

В этом пункте будут продемонстрированы результаты тестирования методами «Черного ящика» (Тест 1) и «Белого ящика» (Тест 2)

**Тест 1. Открытие окна**

Действия:

нажать на кнопку «Помощь», находящуюся в главном меню.

Ожидаемый результат:

открытие дополнительного окна помощи.

Результаты теста:

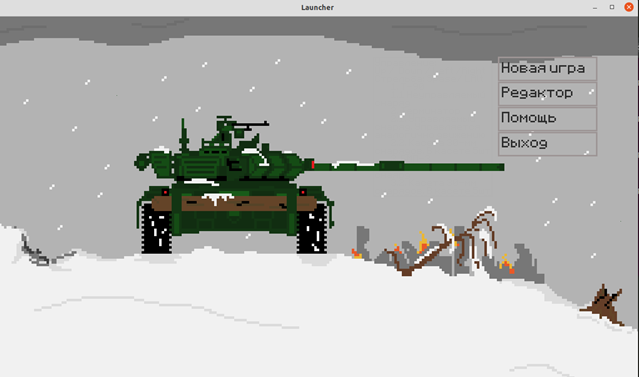


Рисунок 18. Выбор пункта «Помощь»



Рисунок 19. Результат

**Тест 2. Изменение текстуры «Танчиков» противника**

Действия:

открыть файл mainGame.cpp, перейти к 189 строчек и изменить второй аргумент со значения 0 на 40.

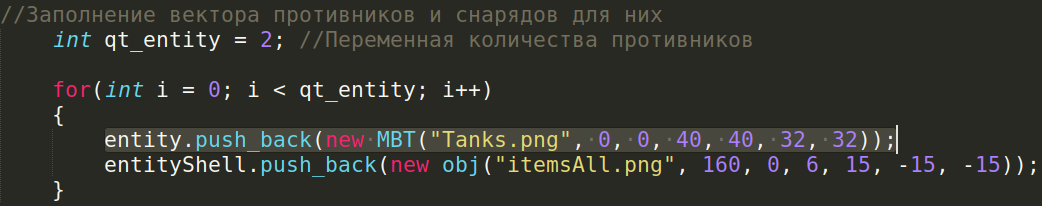


Рисунок 20. Код до изменений

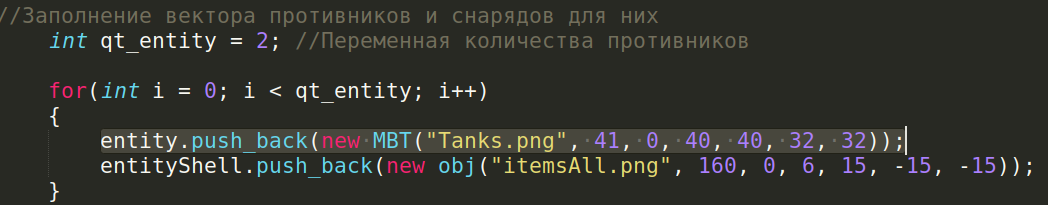


Рисунок 21. Код после изменений

Результат до изменений:

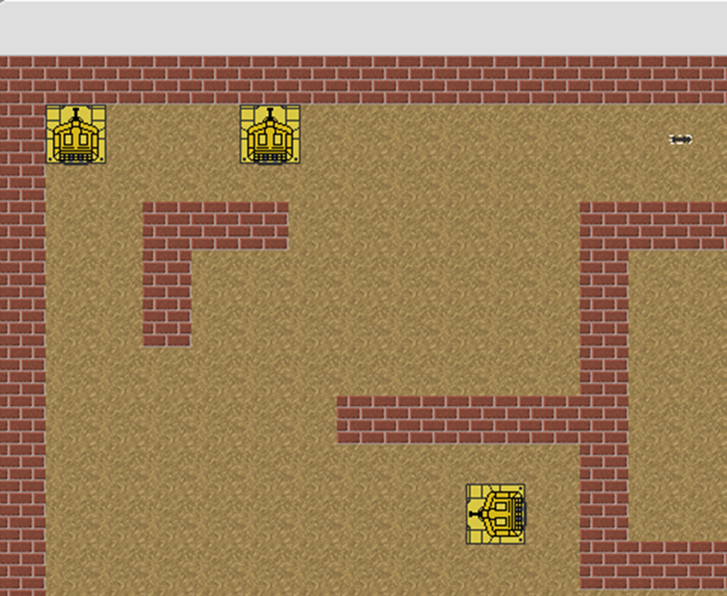


Рисунок 22. Текстуры до изменений

Результат после изменений:

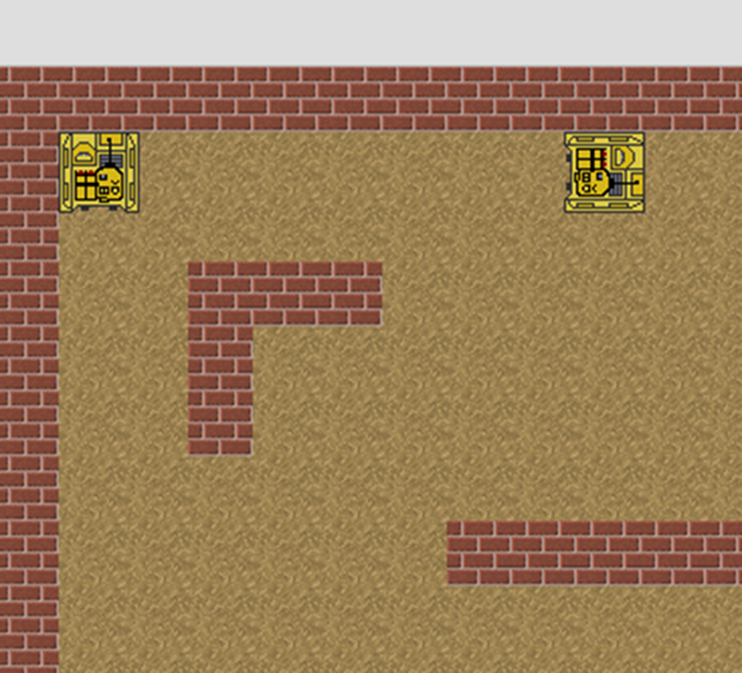


Рисунок 23. Текстуры после

* 1. Описание применения средств отладки

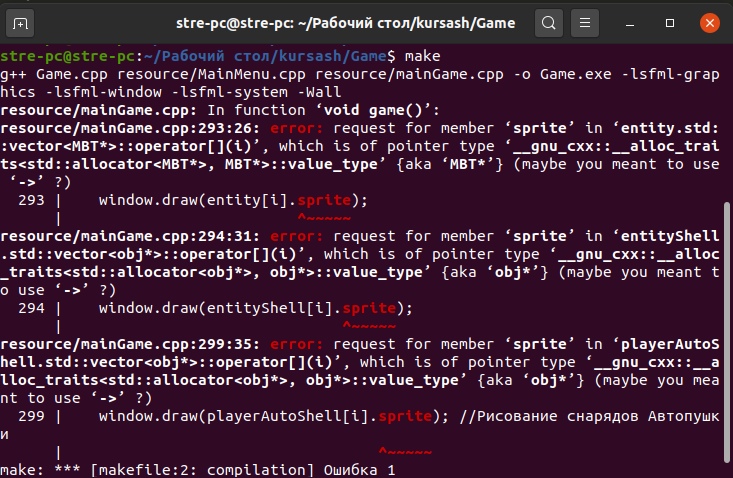
В этом пункте будет описано решение проблемы, возникшее при написании курсового проекта. В ходе написания курсового проекта при попытки запустить один из модулей было получено сообщение об ошибке: 

Рисунок 24. Сообщение об ошибке

После получения данного сообщения были просмотрены 293, 294 и 299 строчки проекта. Ошибка была обнаружена и успешно устранена:

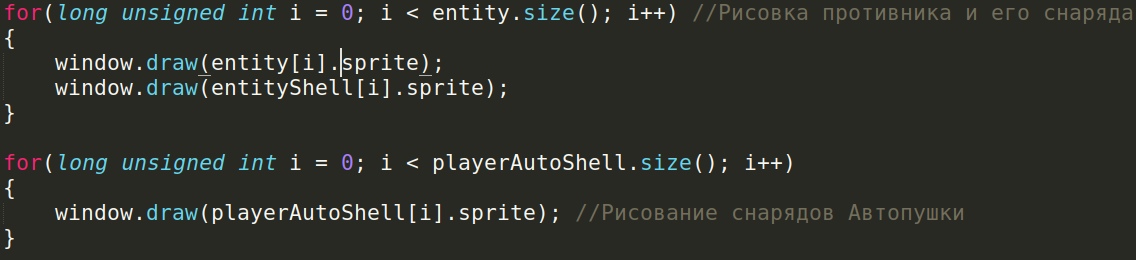


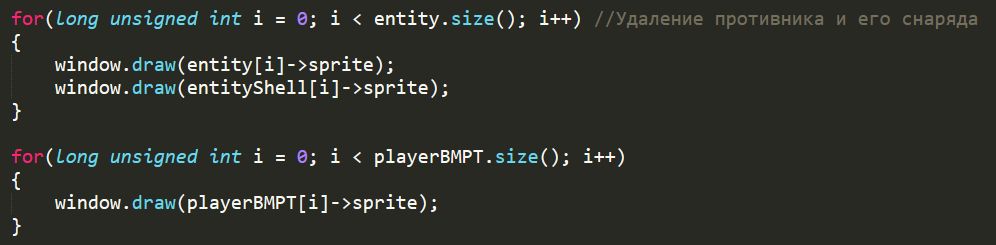
Рисунок 24. Код до решения ошибки

Рисунок 25. Код после решения ошибки

После устранения ошибки приложение было успешно запущено без ошибок и предупреждений



Рисунок 25. Успешный запуск после устранения ошибки

* 1. Анализ оптимальности использования памяти и быстродействия

В данном разделе будет проведен анализ оптимальности использования памяти и быстродействия программы.

Самая большая функция game() в модуле mainGame.cpp имеет 3 вложенных цикла, следовательно, её сложность O(n^3).

Проведя тест по скорости выполнения, было вычислено время выполнения функции game(), В среднем оно составляет 0.55416 микросекунд.

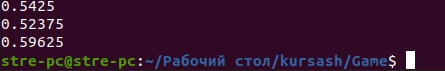


Рисунок 26. Время выполнения главной функции game()

1. Эксплуатационная часть
   1. Руководство оператора
      1. Назначение программы

**Функциональное назначение программы**

Основной функцией проекта «Танчики» является развлекательная. Приложение помогает приятно провести время и расслабиться.

**Эксплуатационное назначение программы**

Основное назначение проекта «Танчики» – показать методы работы с текстурами, спрайтами и текстом, реализовать связь между созданными объектами и действиями игрока

**Состав функций**

**Функция начала новой игры**

Эта функция позволяет начать новую игру

**Функция помощи**

Эта функция позволяет открыть и просмотреть окно помощи

**Функция выбора «Танчика»**

Эта функция позволяет выбрать модель «Танчика»

**Функция игрового процесса**

Эта функция позволяет начать играть в игру

* + 1. Условия выполнения программы

**Минимальный состав аппаратных средств**

Минимальный состав используемых аппаратных средств:

* Windows 7, Windows 8/8.1, Windows 10, Linux Ubuntu 16.04 – 20.04;
* Процессор с тактовой частотой 1 ГГц и выше;
* 2 ГБ ОЗУ и выше;
* Наличие свободного места на жестком диске не меньше 20 МБ;
* Видеоадаптер 20МБ и выше.

**Минимальный состав программных средств**

Системные программные средства, используемые проектом «Танчики», должны быть представлены локализованной версией операционной системы Windows 7, Windows 8/8.1, Windows 10, Linux Ubuntu 16.04 – 20.04

Также для функционирования проекта “Mario” на ПК необходимо предустановленное программное обеспечение стороннего разработчика.

Для Windows систем – IDE CodeBlocks с настроенной библиотекой SFML-2.5.1.

Для Linux систем – установленные репозитории компилятора gcc и SFML-2.5.1.

**Требование к персоналу (Пользователю)**

Конечный пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы или терминалом.

* + 1. Выполнение программы

**Загрузка и запуск программы**

Загрузка и запуск проекта «Танчики» осуществляется в следующем порядке.

**Для Linux-систем:**

распаковать архив с проектом в любое место.

открыть терминал и выполнить команду sudo apt-get install make;

перейти в директорию с проектом;

при первом запуске выполнить команду make installation;

при последующих запусках выполнять make launch.

**Для Windows-систем:**

распаковать архив с проектом в любое место;

скачать SFML с официального сайта (Рис.27);



Рисунок 27. SFML для CodeBlocks

после скачивания распаковать архив в любое место;

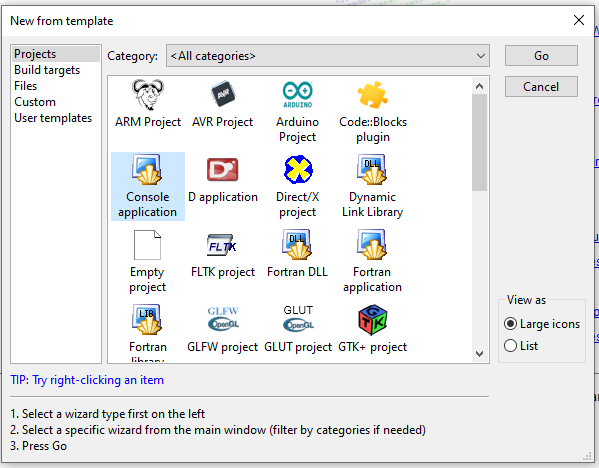
создать проект Console application C++ в IDE CodeBlocks; 

Рисунок 28. Создание проекта

перейти во вкладки Project, далее Build option;

на вкладке «Search directories» проекта добавьте:

* Путь к заголовкам *SFML* ( <sfml-install-path> / include ) к каталогам поиска компилятора(Рис.29)
* Путь к библиотекам *SFML* ( <sfml-install-path> / lib ) к поисковым каталогам компоновщика(Рис.29)

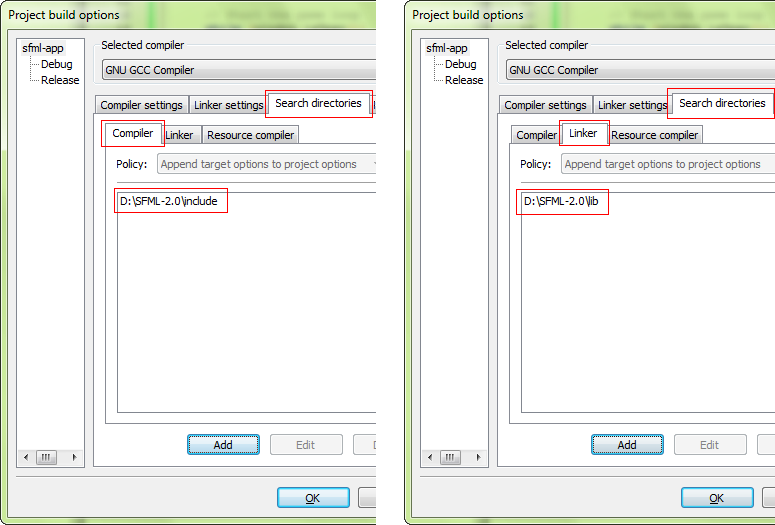


Рисунок 29. Путь к заголовкам и библиотекам

добавьте в «Linker settings» все необходимые библиотеки SFML «sfml-graphics», «sfml-window» и «sfml-system»;

переместите все файлы игры в папку с проектом;

скопирует все файлы из SFML/bin в папке с проектом (Рис.30);

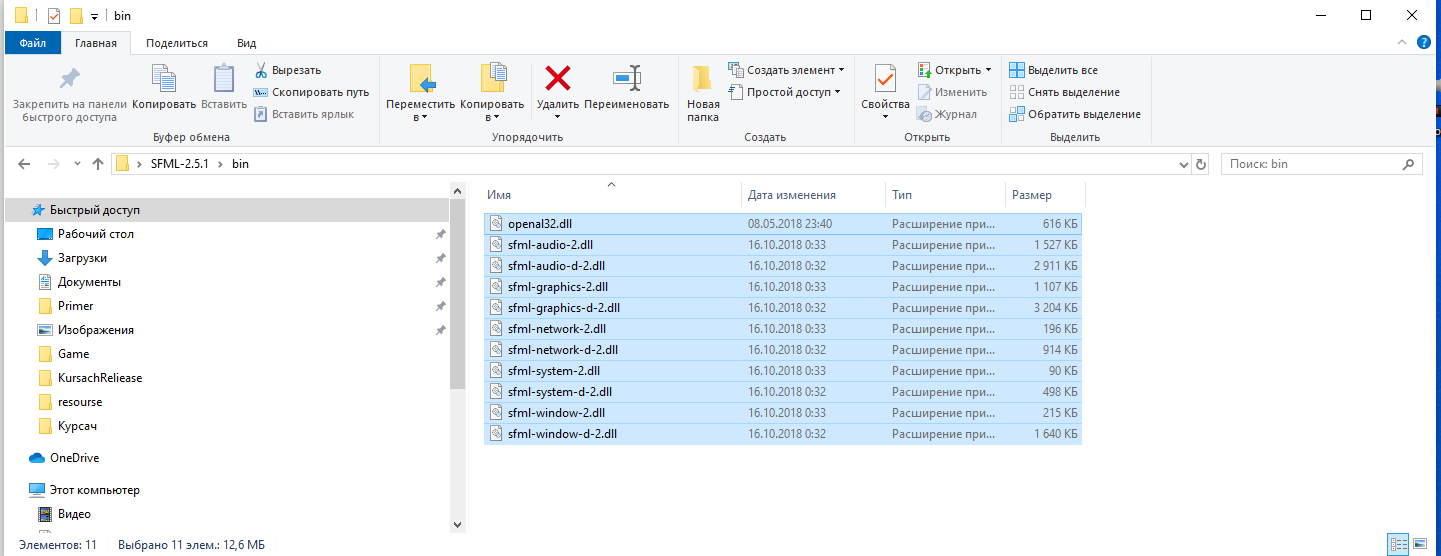


Рисунок 30. Копирование всех файлов

скопируйте весь код из game.cpp в main.cpp проекта;

соберите и запустите проект.

После запуска проекта откроется окно главного меню:



Рисунок 31. Главное меню

Кнопки имеют следующие функции:

Новая игра – служит для открытия окна выбора «Танчика», после запуска игры;

Редактор – неактивная. Добавление функционала планируется в следующих версиях;

Помощь – открывает окно помощи. При повторном нажатии закрывает его

Выход – выход из программы

* 1. Выполнение программы
     1. Выполнение функции начала новой игры

После выбора пункта новой игры откроется окно выбора «Танчика».

* + 1. Функция выбора «Танчика»

В окне предложен выбор «Танчика» для начала игры (Рис.32).

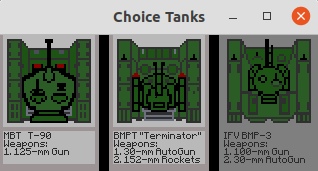


Рисунок 32. Выбор «Танчика»

После выбора откроется окно с игровым процессом.

* + 1. Функция игрового процесса

Представляет собой начало игры. В ней надо управлять «Танчиком» и уничтожать противников (Рис.33). Управление описано в окне помощи.

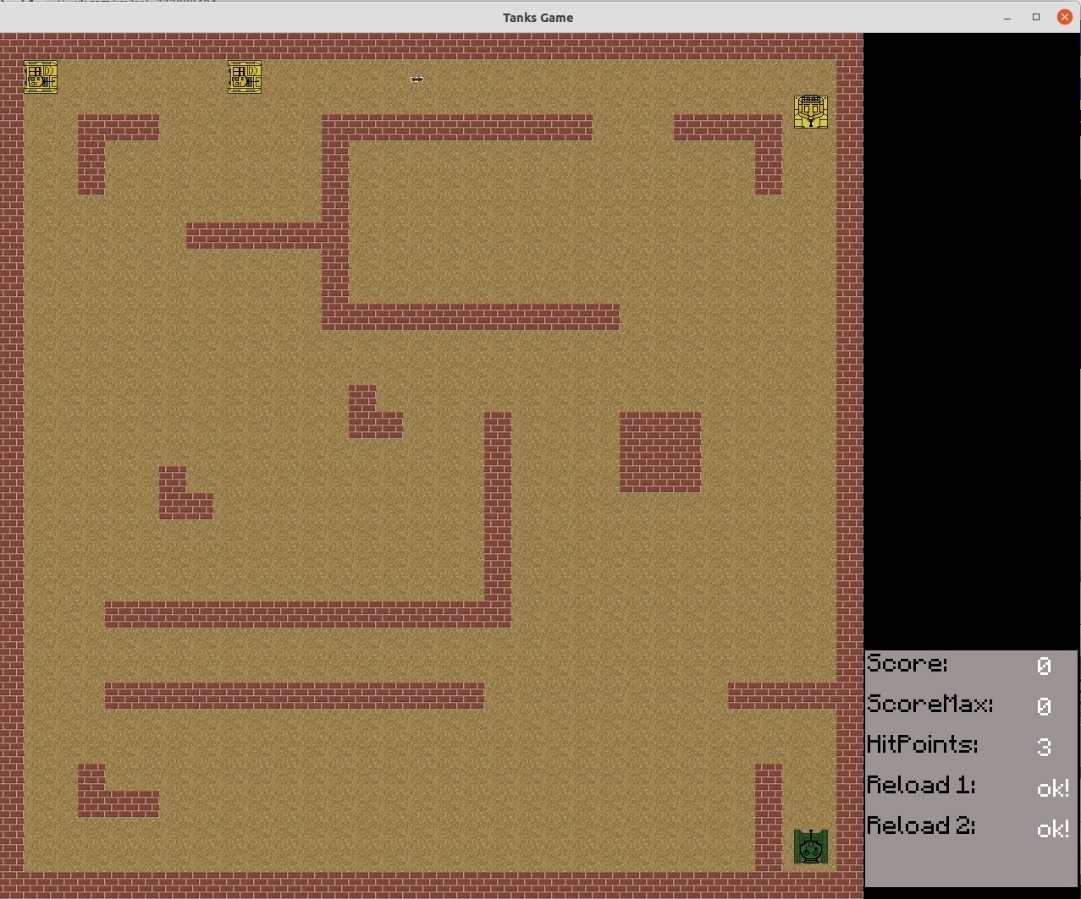


Рисунок 33. Игровой процесс

* + 1. **Функция помощи**

Открывает окно помощи, в котором описано управления «Танчиком» и его вооружением (Рис.34).



Рисунок 34. Окно помощи

* 1. Завершение программы

Для завершения работы программы, если она запущена в «Меню», можно нажать на кнопку «Выход», или на «красный крестик» в правом верхнем углу. Если уже идет игровой процесс, также через «красный крестик».

Заключение

В результате выполнения курсового проекта была написана игра «Танчики» для развлечения и приятного времяпрепровождения.

В ходе работы были проанализированы предметная область, существующие разработки, получены практические навыки по созданию игр на основе библиотеки SFML.

Планируется продолжать работу над данным проектом с целью расширения функционала и игровых механик.

To-do лист:

1. Добавление возможности создания собственных карт;
2. Добавление новых механик и возможностей игроку и противникам;
3. Создание режима для 2 игроков для игры по сети или за одним компьютером.

Список литературы и интернет-источников

1. Описание аркад:

<https://dic.academic.ru/dic.nsf/ruwiki/159579>

1. Игра Battle City:

<https://playminigames.ru/game/tanchiki>

1. Официальный сайт игры Armored Warfare:

<https://armata.my.games/>

1. Официальный сайт SFML:

<https://www.sfml-dev.org/>

1. Официальный сайт Sublime Text 3:

<https://www.sublimetext.com/3>

1. Официальный сайт GNU GCC

<https://gcc.gnu.org/>

Приложение 1. Модуль Game.cpp.

#include <iostream>

#include <vector>

#include <cstring>

#include <sstream>

#include <SFML/Graphics.hpp>

#include "resourse/MainMenu.hpp"

using namespace std;

using namespace sf;

int main()

{

//Вызов фунции лаунчера

launcher();

return 0;

}

Приложение 2. Модуль MainMenu.hpp.

#pragma once

void launcher();

Приложение 3. Модуль MainMenu.cpp.

#include "ButtonClass.hpp"

#include "mainGame.hpp"

//Подсветка кнопок

//Приниается вектор всех кнопок и вектор координат курсора

void press(vector<Button\*> v, Vector2f pos)

{

//i = 1, кроме 1 элемента, это фон

for(long unsigned int i = 1; i < v.size(); i++)

{

//Если мышка неведена на кнопку, то она подсвечивается черным

//(!= 5) кроме 5 элемента, это окно помощи

if(v[i]->sprite.getGlobalBounds().contains(pos.x, pos.y) && i != 5)

v[i]->sprite.setColor(Color::Black);

else if (i != 5)

v[i]->sprite.setColor(Color::White);

}

}

//Смена кадров фона

//Принимется фон окна, время, номер кадра

void FrameMenu(Button\* menu, float time, float &CurrentFrame)

{

CurrentFrame += 0.005 \* time;

if(CurrentFrame > 3)

CurrentFrame = 0;

menu->sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) \* 1280), 0, 1280, 720));

}

//Основная фунция главного меню

void launcher()

{

RenderWindow window(VideoMode(1280, 720), "Launcher"); //создание окна 1280х720

float CurrentFrame = 0; //Номер кадра

bool isOpen = true; //открыто ли окно помощи

Clock clock;

vector<Button\*> mass; //вектор кнопок

mass.push\_back(new Button("Main\_menu.png", 0, 0, 1280, 720, 0, 0)); //Фон [0]

mass.push\_back(new Button("Buttons.png", 2, 2, 200, 50, 1000, 80)); //Новая игра [1]

mass.push\_back(new Button("Buttons.png", 2, 52, 200, 50, 1000, 130)); //Редактор [2]

mass.push\_back(new Button("Buttons.png", 2, 102, 200, 50, 1000, 180)); //Помощь [3]

mass.push\_back(new Button("Buttons.png", 2, 152, 200, 50, 1000, 230)); //Выход [4]

mass.push\_back(new Button("Buttons.png", 202, 2, 240, 280, 750, 80)); //Окно помощи[5]

mass[5]->sprite.setColor(Color(0, 0, 0, 1)); //Покраска окна помощи в прозрачный

//===================================================// Ниже самое важное, малоизменяемое

while(window.isOpen())

{

Vector2i pixelPos = Mouse::getPosition(window);//забираем коорд курсора

Vector2f pos = window.mapPixelToCoords(pixelPos);//переводим их в игровые (уходим от коорд окна)

srand(time(NULL));

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

clock.restart();

time = time / 800;

Event close;

while(window.pollEvent(close))

{

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

window.close();

//==========================================// Тут выбор пункта меню

else if(Mouse::isButtonPressed(Mouse::Left))

{

if(mass[1]->sprite.getGlobalBounds().contains(pos.x, pos.y)) //Если нажата "Новая игра" закрываем окно лайнчера и запускаем игру

{

window.close();

game();

}

if(mass[3]->sprite.getGlobalBounds().contains(pos.x, pos.y)) //Если нажата "Помощь" то, окно помощи становится видимым

{

if(isOpen == false)

{

mass[5]->sprite.setColor(Color(0, 0, 0, 1)); //при повторном нажатии окно становится невидимым

isOpen = true;

}

else

{

mass[5]->sprite.setColor(Color(255, 255, 255, 255));

isOpen = false;

}

}

if(mass[4]->sprite.getGlobalBounds().contains(pos.x, pos.y)) //При нажатии выход, лаунчер закрывется и ничего не происходи

{

window.close();

}

}

}

//===================================================// Выше самое важное, малоизменяемое

press(mass, pos);

FrameMenu(mass[0], time, CurrentFrame);

//==================================================// Рисуем всё, что есть

window.draw(mass[0]->sprite);

for(long unsigned int i = 1; i < mass.size(); i++)

{

window.draw(mass[i]->sprite);

}

window.display();

}

}

Приложение 4. Модуль mainGame.hpp.

#pragma once

void game();

Приложение 5. Модуль mainGame.cpp.

#include "ClassMBT.hpp"

#include "ClassBMPT.hpp"

#include "ClassIFV.hpp"

#include "ClassOBJ.hpp"

//Подсветка кнопок

//Приниается вектор всех кнопок и вектор координат курсора

void press(vector<obj\*> v, Vector2f pos)

{

for(long unsigned int i = 0; i < v.size(); i++)

{

if(v[i]->sprite.getGlobalBounds().contains(pos.x, pos.y))

v[i]->sprite.setColor(Color(0, 0, 0, 1));

else

v[i]->sprite.setColor(Color::White);

}

}

//Отрисовка карты

//Принимается объект карты, граница по х, граница по у

void MapFunct(MapClass \*MapObjF, int i , int j)

{

if(TileMap[i][j] == '0') MapObjF->sprite.setTextureRect(IntRect(0, 0, 32, 32));

if(TileMap[i][j] == ' ') MapObjF->sprite.setTextureRect(IntRect(32, 0, 32, 32));

MapObjF->sprite.setPosition(j \* 32, i \* 32);

}

//Выбор Танчика

//Принимается переменная выбора

void choice(int &a)

{

RenderWindow window(VideoMode(320, 136), "Choice Tanks"); //Рисуем отдельное окно

Clock clock;

vector<obj\*> choicePlayer;

choicePlayer.push\_back(new obj("itemsAll.png", 0, 187, 100, 136, 0, 0));

choicePlayer.push\_back(new obj("itemsAll.png", 100, 187, 100, 136, 110, 0));

choicePlayer.push\_back(new obj("itemsAll.png", 200, 187, 100, 136, 220, 0));

while(window.isOpen())

{

Vector2i pixelPos = Mouse::getPosition(window);//забираем коорд курсора

Vector2f pos = window.mapPixelToCoords(pixelPos);//переводим их в игровые (уходим от коорд окна)

srand(time(NULL));

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

clock.restart();

time = time / 800;

Event close;

while(window.pollEvent(close))

{

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

window.close();

else if(Mouse::isButtonPressed(Mouse::Left))

{

if(choicePlayer[0]->sprite.getGlobalBounds().contains(pos.x, pos.y))

{

a = 1;

window.close();

}

if(choicePlayer[1]->sprite.getGlobalBounds().contains(pos.x, pos.y))

{

a = 2;

window.close();

}

if(choicePlayer[2]->sprite.getGlobalBounds().contains(pos.x, pos.y))

{

a = 3;

window.close();

}

}

}

press(choicePlayer, pos); //Вызываем фунцию подсветки

//Рисуем все элемениы

for(long unsigned int i = 0; i < choicePlayer.size(); i++)

window.draw(choicePlayer[i]->sprite);

window.display();

}

}

//Фунция готовности к стрельбе

//Принимается переменная перезарядки

string isFire(bool isReloadingF)

{

string s = "ok!";

if(isReloadingF == true)

s ="...";

return s;

}

//Фунция записи очков, максимума очков, перезарядки

//Принимается объект текста, очки, максимальные очки, кол-во жизней, переменная перезарядки 1 и 2 вида оружия

void Score(Text \*text, int ScoreF, int \*MaxScoreF, float HitPointF, bool isReloadingF, bool isAutoReloadingF)

{

ostringstream Scr;

if(ScoreF > \*MaxScoreF)

\*MaxScoreF = ScoreF;

Scr << ScoreF << "\n\n" << \*MaxScoreF << "\n\n" << HitPointF << "\n\n" << isFire(isReloadingF) << "\n\n" << isFire(isAutoReloadingF)<< "\n\n";

text->setString(Scr.str());

text->setPosition(1230, 735);

}

//Основная функция игры

void game()

{

//Объект текста для вывода данных на экран

Font font;

font.loadFromFile("resourse/9303.ttf");

Text text;

text.setFont(font);

text.setCharacterSize(24);

text.setFillColor(Color::White);

MapClass MapObj; //Объект карты

Tanks \*player; //Объект игрока

obj\* playerShell; //Объект снаряда игрока

obj\* playerInf = new obj("Buttons.png", 443, 2, 251, 280, 1026, 730); //Окно основной информации

vector<obj\*> playerBMPT; //Объект 30-мм снарядов игрока

obj\* fireAutoGun = new obj("itemsAll.png", 160, 20, 10, 10, 300, 300); //Объект эффекта попадания от АвтоПушки

obj\* fireGun = new obj("itemsAll.png", 173, 21, 15, 15, 400, 400); //Объект эффекта попадания от снаряда

int i; //Переменная выбора танчика

int MaxScore = 0; //Переменная максималных очкнов

int Respawn = 0; //Переменная спавна противника

float TimeResp = 0; //Переменная времени спавна противника

float TimeFire = 0; //Переменная времени стрельбы противником

choice(i); //Вызов фунции выбора

//===================================================//Создание танчика по выбору игрока

if(i == 1)

{

MBT\* ptanks = new MBT("Tanks.png", 0, 82, 40, 40, 942, 942);

obj\* pShell = new obj("itemsAll.png", 160, 0, 6, 15, -15, -15);

player = ptanks;

playerShell = pShell;

}

else if(i == 2)

{

BMPT\* ptanks = new BMPT("Tanks.png", 41, 82, 40, 40, 942, 942);

obj\* pShell = new obj("itemsAll.png", 167, 0, 6, 18, -15, -15);

player = ptanks;

playerShell = pShell;

for(int i = 0; i < 5; i++)

playerBMPT.push\_back(new obj("itemsAll.png", 172, 0, 4, 6, 300, 300));

}

else if(i == 3)

{

IFV\* ptanks = new IFV("Tanks.png", 82, 82, 40, 40, 942, 942);

obj\* pShell = new obj("itemsAll.png", 160, 0, 6, 18, -15, -15);

player = ptanks;

playerShell = pShell;

for(int i = 0; i < 5; i++)

playerBMPT.push\_back(new obj("itemsAll.png", 172, 0, 4, 6, 300, 300));

}

//MBT player("Tanks.png", 0, 82, 40, 40, 942, 942); //MBT SSSR

//obj playerShell("itemsAll.png", 160, 0, 4, 12, -15, -15); //Снаряд

//obj playerShell("itemsAll.png", 164, 0, 4, 12, 300, 300); //ПТУР

//obj playerShell("itemsAll.png", 168, 0, 2, 4, 300, 300); // 30-мм Снаряд

//MBT player("Tanks.png", 82, 82, 40, 40, 300, 300); //IFV SSSR

//Tanks player("Tanks.png", 41, 82, 40, 40, 300, 300); //BMPT SSSR

//Tanks player("Tanks.png", 0, 0, 40, 40, 300, 300); //MBT USA

//Tanks player("Tanks.png", 41, 0, 40, 40, 300, 300); //IFV USA

//Tanks player("Tanks.png", 82, 0, 40, 40, 300, 300); //SuperMBT USA

//===================================================// Создание противников

vector<MBT\*>entity; //Вектор противников

vector<obj\*>entityShell; //Вектор снарядов противника

for(int i = 0; i < 1; i++)

{

entity.push\_back(new MBT("Tanks.png", 0, 0, 40, 40, 32, 32));

entityShell.push\_back(new obj("itemsAll.png", 160, 0, 6, 15, -15, -15));

}

//===================================================//

RenderWindow window(VideoMode(1280, 1024), "Tanks Game"); //Создание окна

Clock clock; //Объект времени

while(window.isOpen()) //Основное окно событий

{

srand(time(NULL));

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

clock.restart();

time = time / 800;

Event close;

while(window.pollEvent(close))

{

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

window.close();

}

//===================================================// Выше самое важное, малоизменяемое

//Вызов методов управления игрока

player->control(time);

player->update(time);

player->interactionWithMap();

player->fire(playerShell, time, playerBMPT);

//===================================================// ИИ

//Спавн нового противника каждые 6 секунд

TimeResp += time;

if(TimeResp >= 8000)

{

Respawn = 1 + rand() % 3;

if(Respawn == 1)

entity.push\_back(new MBT("Tanks.png", 0, 0, 40, 40, 32, 32));

if(Respawn == 2)

entity.push\_back(new MBT("Tanks.png", 0, 0, 40, 40, 942, 32));

if(Respawn == 3)

entity.push\_back(new MBT("Tanks.png", 0, 0, 40, 40, 32, 942));

entityShell.push\_back(new obj("itemsAll.png", 160, 0, 6, 15, -15, -15));

TimeResp = 0;

}

//Стрельба касетных снарядов автопушки игрока каждые полсекунды

TimeFire += time;

if(TimeFire >= 500)

{

fireGun->sprite.setPosition(-15, -15);

fireAutoGun->sprite.setPosition(-15, -15);

TimeFire = 0;

}

//Вызов методов управления для противника

for(long unsigned int i = 0; i < entity.size(); i++)

{

entity[i]->Artificial\_Intelligence(time);

entity[i]->fire(entityShell[i], time, playerBMPT);

entity[i]->update(time);

entity[i]->interactionWithMap();

entity[i]->dead(playerShell, 942, 32, player->damage, playerBMPT, player->AutoDamage, fireAutoGun, fireGun);

player->dead(entityShell[i], 942, 942, entity[i]->damage, fireGun);

if(entity[i]->isDead == true)

{

entity.erase(entity.begin() + i);

entityShell.erase(entityShell.begin() + i);

player->Score += 5;

}

}

Score(&text, player->Score, &MaxScore, player->HitPoint, player->isReloading, player->isAutoReloading);

//===================================================//Рисование карты

for(int i = 0; i < HEIGHT\_MAP; i++)

{

for(int j = 0; j < WIDHT\_MAP; j++)

{

MapFunct(&MapObj, i, j);

window.draw(MapObj.sprite);

}

}

//==================================================// Рисуем всё, что есть

window.draw(player->sprite);

window.draw(playerShell->sprite);

for(long unsigned int i = 0; i < entity.size(); i++) //Удаление противника и его снаряда

{

window.draw(entity[i]->sprite);

window.draw(entityShell[i]->sprite);

}

for(long unsigned int i = 0; i < playerBMPT.size(); i++)

{

window.draw(playerBMPT[i]->sprite);

}

window.draw(fireGun->sprite);

window.draw(fireAutoGun->sprite);

window.draw(playerInf->sprite);

window.draw(text);

window.display();

}

}

Приложение 6. Модуль ClassTanks.cpp.

#pragma once

#include "ClassOBJ.hpp"

#include "map.hpp"

//Класс общих характеристик и методов танчиков

class Tanks

{

protected:

string File; //Название файла текстуры

Image image;

Texture texture;

float speed = 0.0; //Скорость передвижения всех

float x, y, dx, dy, w, h, fx, fy; //координаты, начальная точки спрайта, конечные, временные для передивжения

bool isEntity = false; //Переменная проверки на игрока или противника

int dir = 0; //Переменная направления танчика

float reloading = 0; //Переменная перезарялки

float FullReload[2] = {}; //Массив времени перезарядки

float AutoReloading = 0; //Переменная полного времени перезарядки АвтоПушки

public:

float damage = 0.0; //Переменная урона обычной пушкой или ПТУРОМ

float AutoDamage = 0.0; //Переменная урона снарядом автопушки

float HitPoint = 3.0; //Кол-во жизней

bool isReloading = false; //Переменная проверки статуса перезарядка

bool isAutoReloading = false; //Переменная проверки статуса перезарядки автопушки

bool isDead = false; //Переменная проверки статуса жизни

int Score = 0; //Переменная очков

Sprite sprite;

//Конструктор танчика, задается спрайт и точка появления

Tanks(string F, float dX, float dY, float W, float H, float X, float Y)

{

File = F;

w = W; h = H;

x = X; y = Y;

dx = dX; dy = dY;

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

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(dx, dy, w, h));

sprite.setPosition(x, y);

}

//Метод управления

//Принимает время

void control(float time)

{

if(Keyboard::isKeyPressed(Keyboard::Up)) //Если нажата стрелочка вверх указывается направление и скорость

{

dir = 3; speed = 0.2;

sprite.setTextureRect(IntRect(dx, dy, 40, 40)); //Перерисовывается танк в соответствии с направлением движения

}

if(Keyboard::isKeyPressed(Keyboard::Down)) //Вниз, аналогично вверх

{

dir = 2; speed = 0.2;

sprite.setTextureRect(IntRect(dx + 40, dy + 40, -40, -40));

}

if(Keyboard::isKeyPressed(Keyboard::Left)) //Влево, аналогично вверх

{

dir = 1; speed = 0.2;

sprite.setTextureRect(IntRect(dx + 40, dy + 40 + 41, -40, -40));

}

if(Keyboard::isKeyPressed(Keyboard::Right)) //Вправо, аналогично вверх

{

dir = 0; speed = 0.2;

sprite.setTextureRect(IntRect(dx, dy + 41, 40, 40));

}

}

//Метод передвижения

//Принимаеися время

void update(float time)

{

switch(dir) //Движение танка исходя из направления танчика

{

case 0: fx = speed; fy = 0; break;

case 1: fx = -speed; fy = 0; break;

case 2: fx = 0; fy = speed; break;

case 3: fx = 0; fy = -speed; break;

}

//для плавного движения пройденное растояние умножаем на время

x += fx \* time;

y += fy \* time;

speed = 0; //если ничего не нажато, то танк останавливается

sprite.setPosition(x, y); //Задание позии танчику

}

//Метод проверки столкновения

void interactionWithMap()

{

for(int i = y / 32; i < (y + h) / 32; i++)

for(int j = x / 32; j < (x + w) / 32; j++)

{

if(TileMap[i][j] == '0')

{

if (fy > 0)//если мы шли вниз,

y = i \* 32 - h;//то стопорим координату игрек персонажа. сначала получаем координату нашего квадратика на карте(стены) и затем вычитаем из высоты спрайта персонажа.

if (fy < 0)

y = i \* 32 + 32;//аналогично с ходьбой вверх. fy<0, значит мы идем вверх (вспоминаем координаты паинта)

if (fx > 0)

x = j \* 32 - w;//если идем вправо, то координата Х равна стена (символ 0) минус ширина персонажа

if (fx < 0)

x = j \* 32 + 32;//аналогично идем влево

}

}

}

//Метод противников

//Принимается время

void Artificial\_Intelligence(float time)

{

isEntity = true;

int randGus = 1 + rand() % 10; //Счётчик случайного выбора направления движения

//Аналогично передвижению игрока

if (randGus == 1 || randGus == 5)

{

dir = 3; speed = 0.2;

sprite.setTextureRect(IntRect(dx, dy, 40, 40));

}

if (randGus == 12 || randGus == 6)

{

dir = 2; speed = 0.2;

sprite.setTextureRect(IntRect(dx + 40, dy + 40, -40, -40));

}

if (randGus == 3 || randGus == 7)

{

dir = 1; speed = 0.2;

sprite.setTextureRect(IntRect(dx + 40, dy + 40 + 41, -40, -40));

}

if (randGus == 4 || randGus == 8)

{

dir = 0; speed = 0.2;

sprite.setTextureRect(IntRect(dx, dy + 41, 40, 40));

}

}

//Перегруженный метод засчитывания попадания

//Принимается объкт снаряда, переменные места возраждения, получаемый урон, вектор объектов автопушки, получаемый урон Автопушки,

//Объект попадания обычного снаряд и объект попадания снаряда автопушки

//Толко для противников

void dead(obj\* Shell, float a, float b, float Fdamage, vector<obj\*> AutoGun, float FdamageA, obj\* fireAutoGun, obj\* fireGun)

{

if(Shell->sprite.getGlobalBounds().intersects(sprite.getGlobalBounds())) //Елси спрайт танчика и снаряда пересеклись засчитано попадание

{

fireGun->sprite.setPosition(Shell->sprite.getPosition().x, Shell->sprite.getPosition().y);//На месте попадания рисуется эффект попадния

Shell->sprite.setPosition(-15, -15); //Попавший снаряд "исчезает" за карту

HitPoint -= Fdamage; //Уменьшение жизней

if(HitPoint <= 0) //Если жизни стали меньше или равны 0 то статус уничтожения становится активным

{

isDead = true;

}

}

for(long unsigned int i = 0; i < AutoGun.size(); i++) //Аналогично со снарядом, только для АвтоПушки

{

if(AutoGun[i]->sprite.getGlobalBounds().intersects(sprite.getGlobalBounds()))

{

fireAutoGun->sprite.setPosition(AutoGun[i]->sprite.getPosition().x, AutoGun[i]->sprite.getPosition().y);

AutoGun[i]->sprite.setPosition(-15, -15);

HitPoint -= FdamageA;

if(HitPoint <= 0)

{

isDead = true;

Score = 0;

}

}

}

}

//Перегруженный метод засчитывания попадания

//Принимается объкт снаряда, переменные места возраждения, получаемый урон, объект попадания обычного снаряда

//Аналогично первому методу

// только для игрока

void dead(obj\* Shell, float a, float b, float Fdamage, obj\* fireGun)

{

if(Shell->sprite.getGlobalBounds().intersects(sprite.getGlobalBounds()))

{

fireGun->sprite.setPosition(Shell->sprite.getPosition().x, Shell->sprite.getPosition().y);

Shell->sprite.setPosition(-15, -15);

HitPoint -= Fdamage;

if(HitPoint <= 0)

{

HitPoint = 3;

//Перенос игрока на место появления

x = a;

y = b;

Score = 0;

}

}

}

//Виртульная фунция для стрельбы

virtual void fire(obj\* shell, float time, vector<obj\*> AutoGun)

{

}

};

Приложение 7. Модуль ClassIFV.cpp.

#pragma once

#include "ClassTanks.hpp"

//Класс Боевой Машины Пехоты (IFV)

//Методы стрельбы: Автопушки и неуправляемого снаряда полностью аналогичны описанным в MBT и в BMPT

class IFV : public Tanks

{

private:

long unsigned int i = 0;

//Метод Автоматической Стрельбы

//В касете находится 5 снарядов с небольшой перезарядкой между залпами.

// После отсртрела 5 снарядов начинается долгая перезарядка всей касеты

// Принимает вектор снарядом и время

// Алгоритм аналогичен стрельбое неуправляемым снарядом, только для 5-ти элементов

void ATG(vector<obj\*> shell, float time)

{

if((Keyboard::isKeyPressed(Keyboard::LAlt) && isMoveATG[i] == false) && isAutoReloading == false)

{

vie[i] = dir;

shell[i]->sprite.setPosition(x + 20, y + 20);

isMoveATG[i] = true;

isAutoReloading = true;

AutoReloading = 0;

FullReload[1] = 300;

i++;

}

if(isAutoReloading == true && AutoReloading < FullReload[1])

{

AutoReloading += time;

}

else

{

if(i == shell.size())

{

FullReload[1] = 3000;

i = 0;

}

else

isAutoReloading = false;

}

cout << i << endl;

for(long unsigned int i = 0; i < shell.size(); i++)

{

if(TileMap[(int)shell[i]->sprite.getPosition().y / 32][(int)shell[i]->sprite.getPosition().x / 32] == '0')

{

isMoveATG[i] = false;

shell[i]->sprite.setPosition(-15, -15);

}

if(isMoveATG[i] == true)

{

switch(vie[i])

{

case 1: shell[i]->sprite.move(-0.5 \* time, 0); shell[i]->sprite.setRotation(270); break;

case 0: shell[i]->sprite.move(0.5 \* time, 0); shell[i]->sprite.setRotation(90); break;

case 3: shell[i]->sprite.move(0, -0.5 \* time); shell[i]->sprite.setRotation(0); break;

case 2: shell[i]->sprite.move(0, 0.5 \* time); shell[i]->sprite.setRotation(180); break;

};

}

}

}

//Метод стрельбы

//Выпускается один снаряд без возможноти управления

//Принимает объект снаряда и время

void gun(obj\* shell, float time)

{

if(((Keyboard::isKeyPressed(Keyboard::Space) || isEntity == true) && isMove == false) && isReloading == false)

{

vie1 = dir;

isMove = true;

shell->sprite.setPosition(x + 20, y + 20);

isReloading = true;

reloading = 0;

}

if(TileMap[(int)shell->sprite.getPosition().y / 32][(int)shell->sprite.getPosition().x / 32] == '0')

{

isMove = false;

shell->sprite.setPosition(-15, -15);

}

if(isReloading == true && reloading < FullReload[0])

{

reloading += time;

}

else

isReloading = false;

if(isMove == true)

{

switch(vie1)

{

case 1: shell->sprite.move(-0.5 \* time, 0); shell->sprite.setRotation(270); break;

case 0: shell->sprite.move(0.5 \* time, 0); shell->sprite.setRotation(90); break;

case 3: shell->sprite.move(0, -0.5 \* time); shell->sprite.setRotation(0); break;

case 2: shell->sprite.move(0, 0.5 \* time); shell->sprite.setRotation(180); break;

};

}

}

public:

bool isMove = false; //Переменная для проверки движения снаряда

vector<bool> isMoveATG; //Вектор проверки движения для каждого 30-мм снаряда

vector<int> vie; //Вектор направления для каждого снаряда

int stv = 0;

int vie1;

float FullReload[2] = {1500, 0}; //Массив для полной перезарядки выстрела и касеты

IFV(string F, float dX, float dY, float W, float H, float X, float Y) : Tanks(F, dX, dY, W, H, X, Y)

{

damage = 1.0;

HitPoint = 3.0;

AutoDamage = 0.5;

isMoveATG.push\_back(false);

}

//Метод стрельбы

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

{

gun(shell, time);

if(AutoGun.size() > isMoveATG.size())

{

for(long unsigned int i = AutoGun.size() - isMoveATG.size(); i < AutoGun.size(); i++)

{

isMoveATG.push\_back(false);

vie.push\_back(0);

}

}

ATG(AutoGun, time);

}

};

Приложение 8. Модуль ClassMBT.cpp.

#pragma once

#include "ClassTanks.hpp"

//Класс Основого Боевого Танка(Main Battle Tank)

class MBT : public Tanks

{

private:

//Метод стрельбы

//Выпускается один снаряд без возможноти управления

//Принимает объект снаряда и время

void gun(obj\* shell, float time)

{

//Если нажат пробел и если не противниик, и если снаряд не в движении, и если не перезарядка

if(((Keyboard::isKeyPressed(Keyboard::Space) || isEntity == true) && isMove == false) && isReloading == false)

{

vie = dir; //снаярд принимает направления танчика

isMove = true; //Снаряд считается теперь в дивжения

shell->sprite.setPosition(x + 20, y + 20); //Перемещается на место вылета из танка

//начало перезарядка

isReloading = true;

reloading = 0;

}

//Проверка на столкновение со стеной

if(TileMap[(int)shell->sprite.getPosition().y / 32][(int)shell->sprite.getPosition().x / 32] == '0')

{

//если столкнулся, то остананавливается и перемещается за карту

isMove = false;

shell->sprite.setPosition(-15, -15);

}

//Перезарядка

if(isReloading == true && reloading < FullReload[0])

{

reloading += time;

}

else

isReloading = false;

//Движени снаряда соотвесутвующее направлению танчика

if(isMove == true)

{

switch(vie)

{

case 1: shell->sprite.move(-0.5 \* time, 0); shell->sprite.setRotation(270); break;

case 0: shell->sprite.move(0.5 \* time, 0); shell->sprite.setRotation(90); break;

case 3: shell->sprite.move(0, -0.5 \* time); shell->sprite.setRotation(0); break;

case 2: shell->sprite.move(0, 0.5 \* time); shell->sprite.setRotation(180); break;

};

}

}

public:

bool isMove = false; //Переменная проверки движения снаряда

int vie; //Переменная направления для снаряда

MBT(string F, float dX, float dY, float W, float H, float X, float Y) : Tanks(F, dX, dY, W, H, X, Y)

{

damage = 1.0; //Установка урона

FullReload[0] = 1500; //Установка времени перезарядки

}

//Метод стрельбы

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

{

gun(shell, time);

}

};

Приложение 9. Модуль ClassBMPT.cpp.

#pragma once

#include "ClassTanks.hpp"

class BMPT : public Tanks

{

private:

long unsigned int i = 0; //Счетчик автоматического огня

//Метод Противотанковой Управляемой Ракеты(ПТУР)

//Принимает ПТУР и время

//Всего 4 ракеты, которые вылетают поочереди и могут управляться игроком

//Аналогичен неуправляемой стрельбе

void ATR(obj\* shell, float time)

{

if(((Keyboard::isKeyPressed(Keyboard::Space) || isEntity == true) && isMove == false) && isReloading == false)

{

//ПТУР вылетает из двух напрвлящих поочереди

//После 4 залпа начинается перезарядка всех 4-х ракет

//Между пусками есть перезарядка

if(stv % 2 == 0)

shell->sprite.setPosition(x + 30, y + 30);

else if(stv % 2 == 1)

shell->sprite.setPosition(x + 10, y + 10);

if(stv == 3)

{

stv = 0;

FullReload[0] = 6000;

}

else

stv++;

isReloading = true;

reloading = 0;

isMove = true;

}

//ПТУР привязан на всем протижении полета к направлению танка.

// При смене направления танка менятся направление полета ПТУРа

switch(dir)

{

case 1: shell->sprite.move(-0.3 \* time, 0); shell->sprite.setRotation(270); break;

case 0: shell->sprite.move(0.3 \* time, 0); shell->sprite.setRotation(90); break;

case 3: shell->sprite.move(0, -0.3 \* time); shell->sprite.setRotation(0); break;

case 2: shell->sprite.move(0, 0.3 \* time); shell->sprite.setRotation(180); break;

};

//Проверка столкновения

if(TileMap[(int)shell->sprite.getPosition().y / 32][(int)shell->sprite.getPosition().x / 32] == '0')

{

isMove = false;

shell->sprite.setPosition(-15, -15);

}

//Перезарялка внутри залпп

if(isReloading == true && reloading < FullReload[0])

reloading += time;

else

{

isReloading = false;

FullReload[0] = 2000;

}

}

//Метод Автоматической Стрельбы

//В касете находится 5 снарядов с небольшой перезарядкой между залпами.

// После отсртрела 5 снарядов начинается долгая перезарядка всей касеты

// Принимает вектор снарядом и время

// Алгоритм аналогичен стрельбое неуправляемым снарядом, только для 5-ти элементов

void ATG(vector<obj\*> shell, float time)

{

if((Keyboard::isKeyPressed(Keyboard::LAlt) && isMoveATG[i] == false) && isAutoReloading == false)

{

vie[i] = dir;

shell[i]->sprite.setPosition(x + 20, y + 20);

isMoveATG[i] = true;

isAutoReloading = true;

AutoReloading = 0;

FullReload[1] = 300;

i++;

}

if(isAutoReloading == true && AutoReloading < FullReload[1])

{

AutoReloading += time;

}

else

{

if(i == shell.size())

{

FullReload[1] = 3000;

i = 0;

}

else

isAutoReloading = false;

}

//Цикл прощета движения и столкновения каждого снаряда

for(long unsigned int i = 0; i < shell.size(); i++)

{

if(TileMap[(int)shell[i]->sprite.getPosition().y / 32][(int)shell[i]->sprite.getPosition().x / 32] == '0')

{

isMoveATG[i] = false;

shell[i]->sprite.setPosition(-15, -15);

}

if(isMoveATG[i] == true)

{

switch(vie[i])

{

case 1: shell[i]->sprite.move(-0.5 \* time, 0); shell[i]->sprite.setRotation(270); break;

case 0: shell[i]->sprite.move(0.5 \* time, 0); shell[i]->sprite.setRotation(90); break;

case 3: shell[i]->sprite.move(0, -0.5 \* time); shell[i]->sprite.setRotation(0); break;

case 2: shell[i]->sprite.move(0, 0.5 \* time); shell[i]->sprite.setRotation(180); break;

};

}

}

}

public:

bool isMove = false;

vector<bool> isMoveATG; //Вектор проверки движения для каждого снаряда

vector<int> vie; //Вектор направления для каждого снаряда

int stv = 0; //Счетчик напрвления из которой вылетит ПТУР

float FullReload[2] = {0, 0}; //Массив для полной перезарядки ПТУРов и касеты

BMPT(string F, float dX, float dY, float W, float H, float X, float Y) : Tanks(F, dX, dY, W, H, X, Y)

{

damage = 2.0;

AutoDamage = 0.5;

isMoveATG.push\_back(false);

}

//Метод стрельбы

//Заполняет вектора вида и проверки движения для АвтоПушки

//Принимает снаряд, время и вектор очереди

void fire(obj\* shell, float time, vector<obj\*> AutoGun)

{

//вызов метода ПТУР

ATR(shell, time);

if(AutoGun.size() > isMoveATG.size())

{

for(long unsigned int i = AutoGun.size() - isMoveATG.size(); i < AutoGun.size(); i++)

{

isMoveATG.push\_back(false);

vie.push\_back(0);

}

}

//вызов метода АвтоПушки

ATG(AutoGun, time);

}

};

Приложение 10. Модуль ClassOBJ.cpp.

#pragma once

#include <iostream>

#include <vector>

#include <cstring>

#include <sstream>

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

//Класс для объектов

//Полностью аналогичен классу Button

class obj

{

public:

float x, y, dx, dy, w, h;

string File;

Image image;

Texture texture;

Sprite sprite;

obj(string F, float dX, float dY, float W, float H, float X, float Y)

{

File = F;

w = W; h = H;

x = X; y = Y;

dx = dX; dy = dY;

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

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(dx, dy, w, h));

sprite.setPosition(x, y);

}

};

Приложение 11. Модуль ButtonsClass.cpp.

#pragma once

#include <iostream>

#include <vector>

#include <cstring>

#include <sstream>

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

//Класс кнопок

class Button

{

public:

float x, y, dx, dy, w, h; //Координаты начальных точек, конечных и расположения спрайта

string File;

Image image;

Texture texture;

Sprite sprite;

Button(string F, float X, float Y, float W, float H, float dX, float dY)

{

File = F;

w = W; h = H;

x = X; y = Y;

dx = dX; dy = dY;

image.loadFromFile("images/" + File); //Загрузка изображения

texture.loadFromImage(image); //Создание техстуры

sprite.setTexture(texture); //Создание спрайта

sprite.setTextureRect(IntRect(x, y, w, h)); // Установка границы рисуемого спрайта

sprite.setPosition(dx, dy); //Задание позиции спрайта

}

};

Приложение 12. Модуль Map.cpp.

#pragma once

using namespace std;

using namespace sf;

//Класс для карты

class MapClass

{

public:

Image image;

Texture texture;

Sprite sprite;

MapClass()

{

image.loadFromFile("images/itemsAll.png");

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(0, 0, 32, 32));

sprite.setPosition(500, 500);

}

};

//Размеры карты

const int HEIGHT\_MAP = 32;

const int WIDHT\_MAP = 32;

//Карта

String TileMap[HEIGHT\_MAP] = {

"00000000000000000000000000000000",

"0 0",

"0 0",

"0 000 0000000000 0000 0",

"0 0 0 0 0",

"0 0 0 0 0",

"0 0 0",

"0 000000 0",

"0 0 0",

"0 0 0",

"0 00000000000 0",

"0 0",

"0 0",

"0 0 0",

"0 00 0 000 0",

"0 0 000 0",

"0 0 0 000 0",

"0 00 0 0",

"0 0 0",

"0 0 0",

"0 0 0",

"0 000000000000000 0",

"0 0",

"0 0",

"0 00000000000000 00000",

"0 0",

"0 0",

"0 0 0 0",

"0 000 0 0",

"0 0 0",

"0 0 0",

"00000000000000000000000000000000"

};