Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Колледж космического машиностроения и технологий

ОТЧЕТ

по производственной практике ПП.01.01 по модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» по специальности 09.02.03 «Программирование в компьютерных системах»

Студент 3 курса группы П2-17

Форма обучения: очная

Стрельников Сергей Дмитриевич

Место прохождения практики

Государственном бюджетном образовательном учреждении высшего образования Московской области «Технологический университет»

(название организации)

Срок прохождения практики с «13» января 2020 г. по «14» марта 2020 г.

Руководители практики

От организации: заведующий мастерской _____ Попов В.Н.

(Должность) (Подпись) (ФИО)

От колледжа: преподаватель _____ Родичкин П.Ф.

(подпись)

Итоговая оценка по практике

_____

Оглавление

Введение

На 3 курсе обучения в ККМТ, студентом группы П2-17 Стрельниковым Сергеем была проведена производственная практика по модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем». Студент получил задание разработать игру.

Во время прохождения практики я поставил для себя следующие цели:

Приобрести опыт работы по специальности.

Закрепить теоретические знания, полученные во время учебы. Проанализировать работы отдела.

Закрепить навыки в разработке проектной и технической документации.

Закрепить навыки отладки и тестирования программных модулей.

Для выполнения вышеупомянутых мной целей я выдвинула следующие задачи:

Изучить специфику деятельности организации.

Установить необходимые инструменты для работы. Найти подходящую литературу.

1. Общие сведения о организации.

1.1. Структура организации характеристика основных видов деятельности.

Данное предприятие работает в сфере образования. Университет образован 16 июля 1998 года в форме некоммерческой организации с названием: Негосударственное образовательное учреждение «Королевская академия управления, экономики и социологии».

Технологический университет (ранее Финансово-технологическая академия; Королевский институт управления, экономики и социологии) создан для подготовки кадров новой информации, воспроизводства интеллектуальных ресурсов, формирования инновационных проектов и технологий. Академия находится в наукограде Королеве Московской области – уникальном центре интеллектуальных ресурсов, которые используются для интеграции важнейших знаний и создания систем глобального масштаба.

20 января 2015 года постановлением Правительства Московской области Академии присвоен статус «университета» и вуз переименован в Государственное бюджетное образовательное учреждение высшего образования Московской области «Технологический университет».

Организационная структура колледжа представлена на Рис. 6.1 в Приложении 1.

1.2. Должностные обязанности оператора ЭВМ, техника – программиста, инженера – программиста.

1.2.1. Должностные обязанности оператора ЭВМ.

осуществляет техническую подготовку документации, необходимой в процессе работы компании. Выполняет копирование документов на ксероксе;

выполняет набор различных текстов с соблюдением правил орфографии и пунктуации, а также стандартов оформления организационно-распорядительной документации;

осуществляет работу с электронной почтой, принимает входящие электронные письма и следит за своевременной отправкой исходящих;

распечатывает и систематизирует нужные документы;

заносит в компьютерные базы данных различную информацию, важную и необходимую для работы компании;

следит за состоянием компьютера и копировальной техники;

своевременно информирует руководство о необходимости приобретения материалов, непосредственно относящихся к производственному процессу.

1.2.2. Должностные обязанности техника – программиста.

выполняет работу по обеспечению механизированной и автоматизированной обработки, поступающей в ВЦ (ИВЦ) информации, разработки технологии решения экономических и других задач производственного и научно-исследовательского характера;
принимает участие в проектировании систем обработки данных и систем математического обеспечения машины;
выполняет подготовительные операции, связанные с осуществлением вычислительного процесса, ведет наблюдение за работой машин;
составляет простые схемы технологического процесса обработки информации, алгоритмы решения задач, схемы коммутации, макеты, рабочие инструкции и необходимые пояснения к ним;
разрабатывает программы решения простых задач, проводит их отладку и экспериментальную проверку отдельных этапов работ;
выполняет работу по подготовке технических носителей информации, обеспечивающих автоматический ввод данных в вычислительную машину, по накоплению и систематизации показателей нормативного и справочного фонда, разработке форм исходящих документов, внесению необходимых изменений и своевременному корректированию рабочих программ;
участвует в выполнении различных операций технологического процесса обработки информации (прием и контроль входной информации, подготовка исходных данных, обработка информации, выпуск исходящей документации и передача ее заказчику);
ведет учет использования машинного времени, объемов выполненных работ;
выполняет отдельные служебные поручения своего непосредственного руководителя.

1.2.3. Должностные обязанности инженера – программиста.

на основе анализа математических моделей и алгоритмов решения экономических и других задач разрабатывает программы, обеспечивающие

возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники, проводит их тестирование и отладку;
разрабатывает технологию решения задач по всем этапам обработки информации;
осуществляет выбор языка программирования для описания алгоритмов и структур данных;
определяет информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля;
выполняет работу по подготовке программ к отладке и приводит отладку; определяет объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;
осуществляет запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач;
проводит корректировку разработанной программы на основе анализа выходных данных;
разрабатывает инструкции по работе с программами, оформляет необходимую техническую документацию;
определяет возможность использования готовых программных продуктов;
осуществляет сопровождение внедрения программ и программных средств;
разрабатывает и внедряет системы автоматической проверки правильности программ, типовые и стандартные программные средства,
составляет технологию обработки информации;
выполняет работу по унификации и типизации вычислительных процессов;
принимает участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектировании программ, позволяющих расширить область применения вычислительной техники.

1.3. Основные функции отдела.

Производственно-технологическая: разработка алгоритма решения задачи на основе предложенной модели; программная реализация алгоритма; отладка и тестирование программных продуктов; модификация программных продуктов; адаптация и настройка программных продуктов; сопровождение программных

продуктов; разработка и эксплуатация баз данных; обеспечение достоверности информации при использовании баз данных;

Организационно-управленческая: организация работы коллектива исполнителей; планирование и организация работ; выбор оптимальных решений при планировании работ в условиях нестандартных ситуаций; участие в оценке качества и экономической эффективности деятельности; обеспечение техники безопасности.

1.4. Документооборот предприятия, структурного подразделения.

Документооборот Отдела в сфере поставленной мне на практике задачи состоит из нескольких этапов:

получение приказа и распределение работы между сотрудниками;

перечень существующих дел в Отделе; годовой план работ;

годовой отчет по проделанной работе.

Вид построенной IDEF модели по плану документооборота представлен на рисунках 1 – 3:

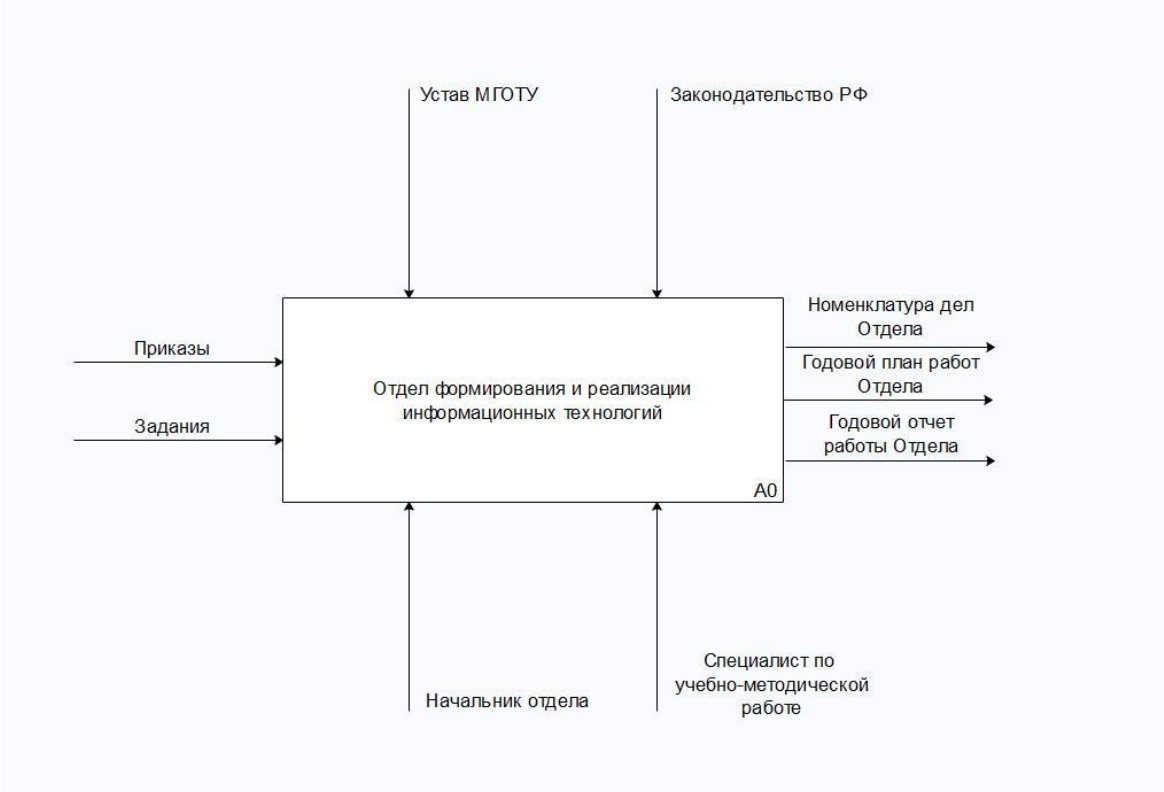


Рис. 1. - IDEF - модель 1 уровень

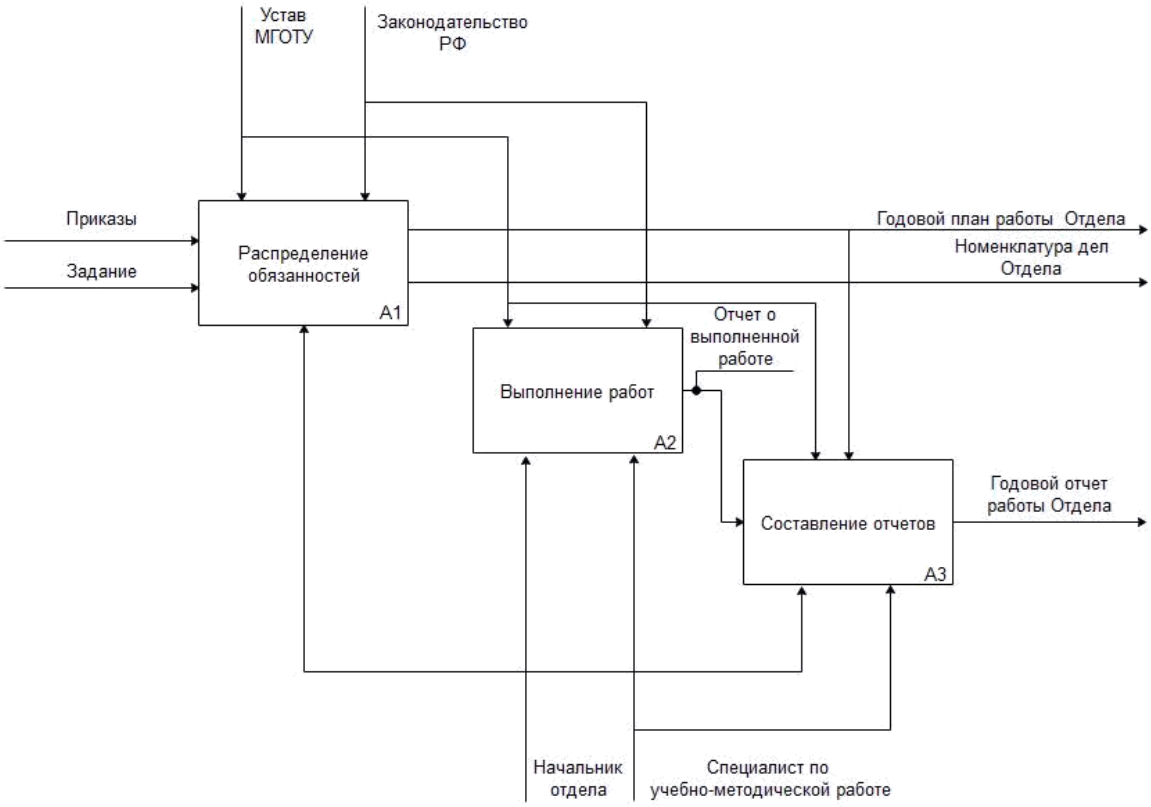

Устав
МГОТУ
Законодательство
РФ
Приказы
Задание
Распределение
обязанностей
A1
Годовой план работы Отдела
Номенклатура дел
Отдела
Отчет о
выполненной
работе
Выполнение работ
A2
Составление отчетов
A3
Годовой отчет
работы Отдела
Начальник
отдела
Специалист по
учебно-методической работе

Рис. 2 - IDEF - модель 2 уровень

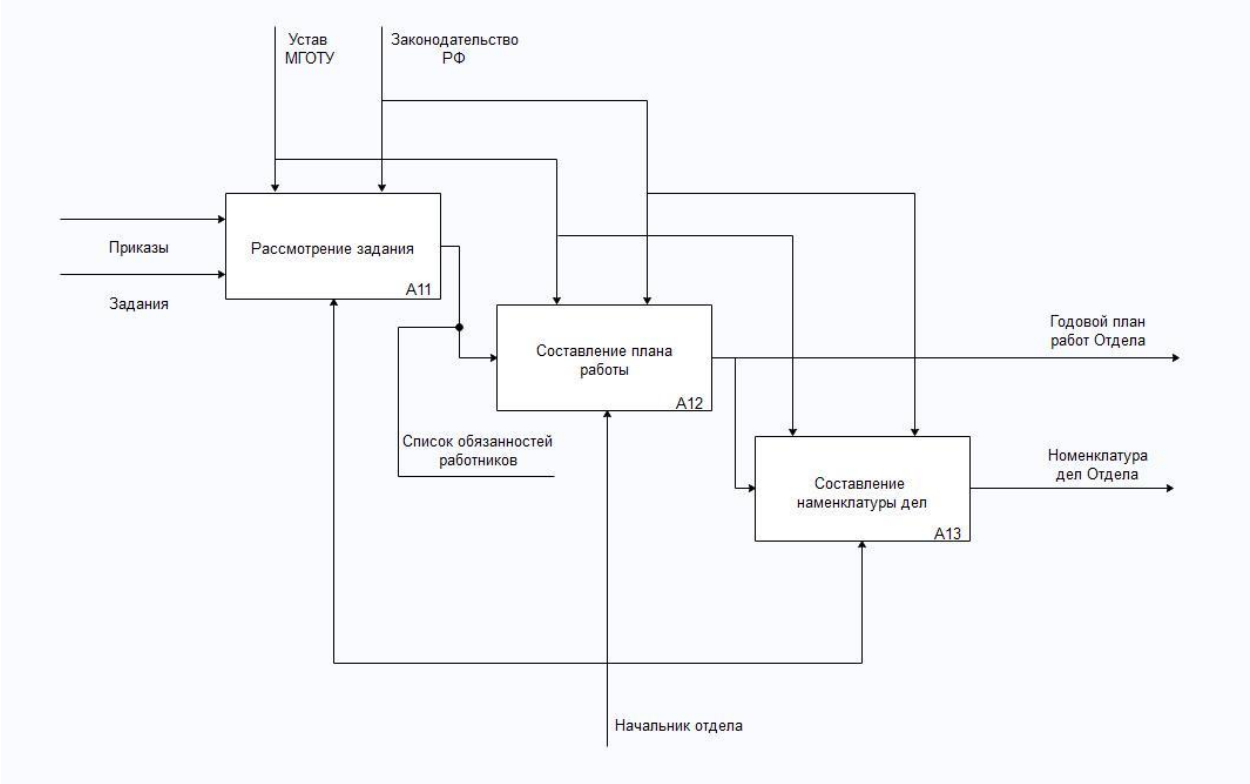


Рис. 3 – IDEF – модель подуровня блока «Распределение задания.

1. Содержание выполняемых видов работ

1.5. Разработка спецификаций отдельных компонентов

Общее задание было разделено на 5 этапов:

Концепция. На основе выданного задания было принято решение писать несколько игр в трех разных жанрах: **2D** – шутер, **2D -**аркада и логическую игру на двух игроков.

Написание основной логики первого режима.

Написание основной логики второго режима.

Написание основной логики третьего режима.

Написание общего кода готового продукта на основе сделанных фрагментов.

1.6. Назначение разработки

Функциональным назначением приложения является сфера развлечения.

Приложение может эксплуатироваться в любых условиях. Не требует специального оборудования и дополнительных установок. Пользователями программы могут быть обычные люди, не имеющие специального образования.

1.7. Выполняемые функции программного продукта

Приложение должно обеспечивать возможность выполнения перечисленных ниже функций:

Функция запуска главного меню.

Функция запуска первого режима.

Функция запуска второго режима.

Функция запуска третьего режима.

Функция работы со звуковыми и музыкальными эффектами, а именно увеличения и уменьшения громкости.

Функция вызова окна информации F.A.Q.

Функция выхода из программы.

1.8. Разработка приложения

Разработанное приложение состоит из 4 разных по функционалу компонента:

2. Первый компонент представлен «Главным меню игры» из которого открывается доступ к остальным функциям приложения (СКРИНШОТ)

   Второй компонент представлен 2D – шутером без сюжета с элементами RPG разрабатывается только в версии для PC. Игровой процесс первого режима представляет собой поиск оружия, поиск аптечек в виде яблок, ведение боевых действий с противниками в виде Уток и в обходе установленных ловушек. Игрок без

ограничений передвигается по игровой карте и занимается доступными в игре действиями. Он может осматривать территорию, собирать яблоки, подбирать и класть оружие в разных местах, отстреливать противников или просто наблюдать за ними, собирать ловушки противника, умирать от них, пополнять свое здоровье яблоками, потраченное на ловушки уток.

Поиск оружия заключается в нахождении игроком автоматической винтовки, без которой невозможно вести боевые действия со стороны игрока.

Поиск аптечек в виде яблок является необязательной частью прохождения игры и заключается в поиске раскиданных по игровой карте яблок и подбором их в и инвентарь. Их можно применить только при потере части здоровья игроком и если игрок жив.

Боевые действия заключаются в стрельбе из автоматической винтовки по противникам, со стороны игрока, и в расстановке ловушек, называемых «Файлы Python», со стороны Уток.

Игрок, потеряв фиксированное количество здоровья становится неиграбельным до тех пор, пока не будет «воскрешен».

3. Третий компонент представлен Аркадой с камерой вида сверху для двух игроков. Игровой процесс представляет собой в управлении боевой единицей, именуемой танком, и ведение боевых действий против танков противника. За каждое уничтожение противника игрок получает очки опыта и увеличении считчика убийств. При уничтожении игрока очки опыта, счетчик убийств обнуляется, и игрок перемещается на место воскрешения и может продолжать игру без ограничений. Есть возможность просмотреть рекорды. Игра рассчитана на двух игроков с возможность игры в одиночку. Игроки по отношению друг к другу являются союзниками и не могу уничтожить друга.

Управление танком заключается в перемещении по карте, стрельбы по противникам, тактическом отступлении при ответном огне и использовании укрытий как часть тактики.

Боевые действия заключаются в попытках вести огонь по вражеским танкам, находящимся на линии огня и в их уничтожении прямым попаданием, а также в необходимости контролировать ситуацию и не попадая под ответный огонь.

Четвертый компонент представлен классической логической игрой в «Крестики – Нолики». Игровой процесс третьего режима заключается в игре двух игроков. Один играет «крестиками», второй – «ноликами». Игра ведется на квадратном поле 3 на 3. Игроки по очереди ставят на свободные клетки поля 3х3 знаки (один всегда

крестики, другой всегда нолики). Первый, выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает. Первый ход делает игрок, ставящий крестики.

3. Тестирование приложения

3.1.1. Тестирование методом черного ящика

Тестирование корректной работы программы методом черного ящика проводятся по следующим сценариям:

Сценарий 1: Первый режим. Подбор предметов. Тестирование главного героя на возможность взаимодействия с предметами.

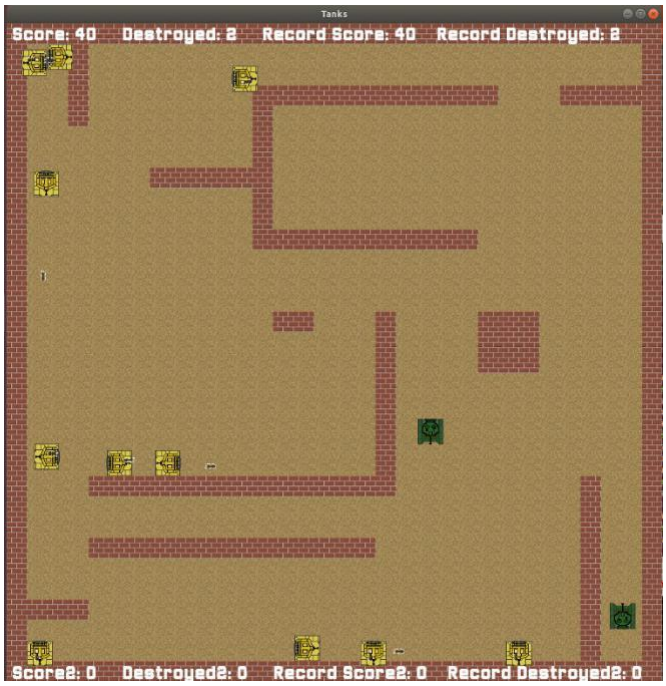
1. Подойти к предмету «Винтовка»

2. Нажать R.

Результат: «Винтовка» переместилась в «руки» главному герою и появилась возможность вести огонь.

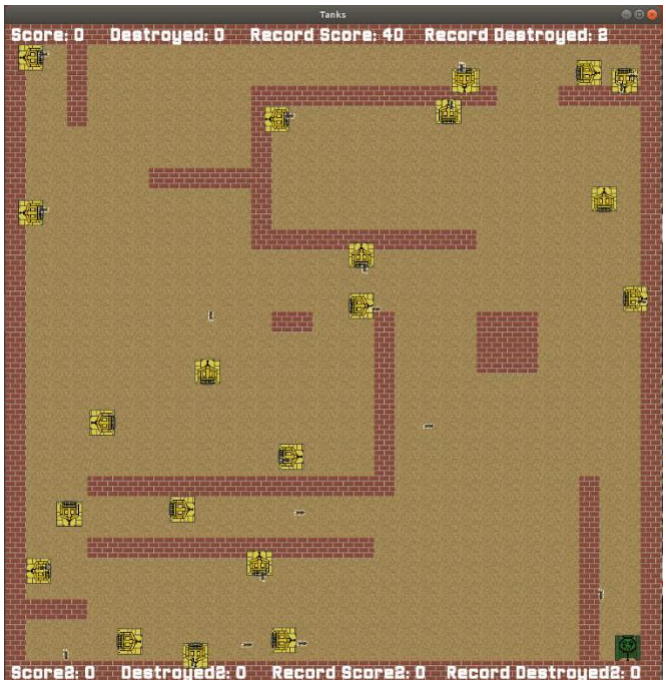
Сценарий 2: Второй режим. Восстановление после уничтожения. Тестирование главного героя на восстановление в установленной точке после попадания снаряда противника и сброса очков и счета уничтоженных противников.

1. Переместиться на линию огня противника.



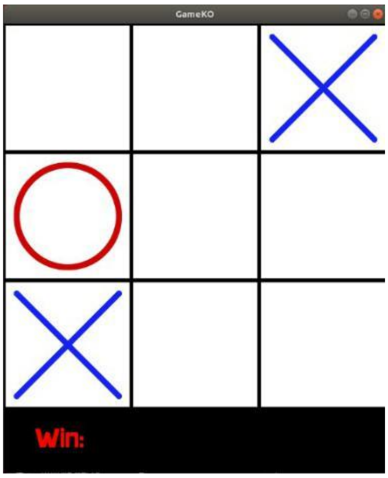
2. Дождаться попадания его снаряда

Результат: перемещение главного героя в нижний правый угол. Сброс счёта очков и уничтоженных противников.

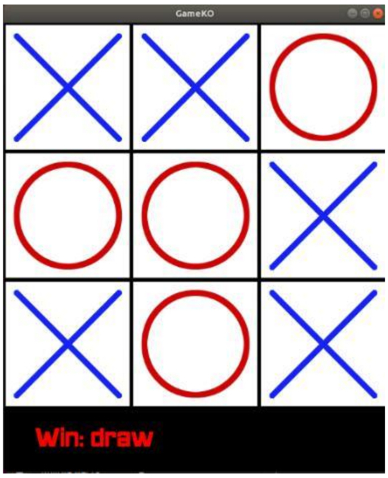


Сценарий 3: Третий режим. Тест на ничью. Тестирование на корректный вывод ничьи.

1. Создать сценарий, при котором по правилам «Крестики - Нолики» будет ничья.



Результат: В поле вывода выводится «draw».



3.2. Отчёт о проведении тестирования

При тестировании программы методом черного ящика по вышеуказанным сценариям прошли успешно.

3.3. Оптимизация программного кода

Пример неоптимизированного кода (BobnevFunct.cpp)

```
vector<Entity>VecEntity;
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100, 1700, 2300));
```

```
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png",  0,  0,  100,  100,  1700,  2300));
VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100, 1700, 2300));
```

Этот же код после оптимизации (BobnevFunct.cpp)

```
vector<Entity>VecEntity;

    for(int i = 0; i < 12; i++)
    {
      VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100, 1700, 2300));
    }
```

3.4. Условия применения приложения

Для работы приложения необходима следующая программно-аппаратная конфигурация:

Ubuntu 16.04 и выше
ОЗУ 2 ГБ и выше
ПЗУ 2 ГБ и выше
Процессор Intel Core i3 и выше
Видеокарта Intel HD Graphics 630 и выше
Любая DirectX 5 совместимая звуковая карта Клавиатура, мышь

3.5. Обращения к приложению

Для запуска приложения необходима сделать следующие действия:

1. Через терминал переместиться в папку с игрой
4. Напечатать make и нажать Enter
5. Если возникли ошибки после нажатия Enter вставить в терминал команду **`Sudo apt-get install make`** и затем повторить пункт 2

6. Вывод

Полученные навыки:

Разработка приложений в системе UNIX на базе библиотеки SFML
Дизайн уровней в библиотеке SFML
Рисование **2D –** моделей в приложении GIMP

Полученные умения

Работа с библиотекой SFML
Работа с терминалом Ubuntu 19.04
Работа в графическом редакторе GIMP

7. Заключение

Перед прохождением производственной практики в Государственном бюджетном образовательном учреждении высшего образования Московской области «Технологический университет» мной были поставлены следующие основные цели:

Приобрести опыт работы по специальности.

Закрепить теоретические знания, полученные во время учебы.

Выполнение требований и действий, предусмотренных программой производственной практики и заданий руководителя.

Закрепить навыки в разработке проектной и технической документации.

Закрепить навыки отладки и тестирования программных модулей.

По окончанию практики я добился, чего хотел. Для достижения своих целей я использовал интернет – источники, документацию средств разработки.

Во время прохождения практики я приобрел опыт работы по специальности. Так же был закреплен навык разработки проектной и технической документации и навык отладки и тестирования программных модулей.

По окончанию практики был составлен отчёт.

8. Дневник практики

Дата	Содержание работы	Отметка о выполнении работы	Подпись руководителя практики
13.01 – 19.01	Обсуждение концепции игры. Выбор языка программирования, выбор среды разработки, выбор программного обеспечения для 2D - рисования		
20.01 – 26.01	Обсуждение и создание игрового мира и основных механик		
27.01 – 2.02	Разработка третьего режима и меню		
3.02 – 16.02	Разработка второго режима		
17.02 – 1.03	Разработка третьего режима		
02.03 – 08.03	Разработка музыкального сопровождения, подбор звуковых эффектов.		
09.03 – 13.03	Оформление отчёта. Обработка листингов, группировка отчёта		

Таблица 1. Дневник практики

9. Список используемых источников

1. https://www.sfml-dev.org/documentation/2.5.1/
2. https://www.sfml-dev.org/tutorials/2.5/
3. https://kychka-pc.ru/category/sfml
4. https://uroki-gimp.ru/

10. Приложения

10.1. Приложение 1

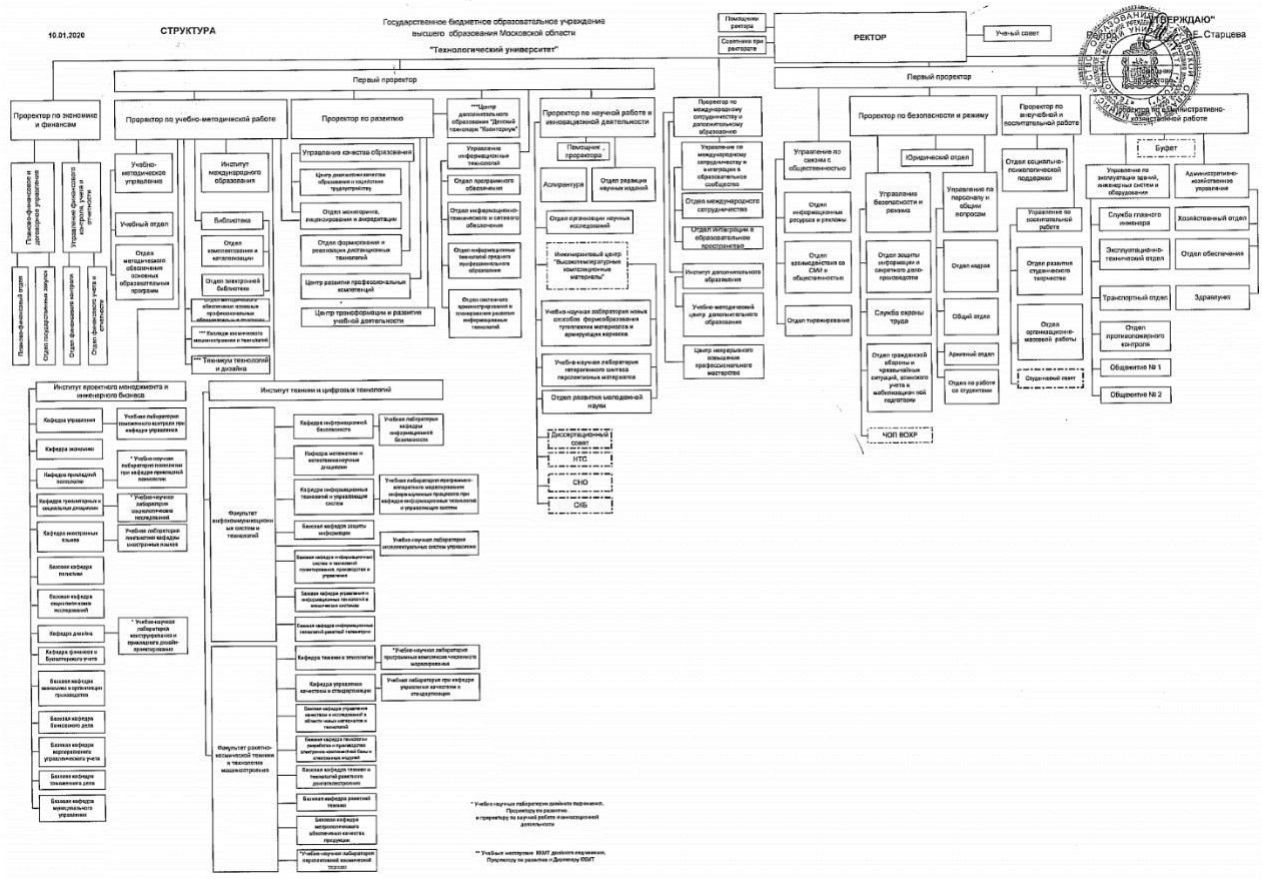


Рисунок 1 Организационная структура колледжа

10.2. Приложение 2.

10.2.1. Листинг 1. Главный файл

10.2.1.1. Bobved.cpp

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <ctime>
#include <cmath>
#include <cstring>
#include <vector>
#include <sstream>

#include "resourse/BobnevFunct.hpp"
#include "resourse/minigame1.hpp"
#include "resourse/menu.hpp"
#include "resourse/TanksGame.hpp"

using namespace std;
using namespace sf;
```

```
int BobnevGame();

int zapucs()
{
	int a = launcher();
	if(a == 10)
		BobnevGame();
	else if(a == 11)
	{
			KrestNol();
			zapucs();
	}
	else if(a == 12)
	{
		tanks();
		zapucs();
	}

	return 0;
}

int main()
{
	zapucs();
	return 0;
}
```

10.2.2. Листинг 2 - 5. Первый режим

10.2.2.1. BobnevFunct.hpp

```
#include "classPlayer.hpp"

using namespace std;
using namespace sf;

int BobnevGame()
{
	RenderWindow window(VideoMode(1200, 800),
	"BobnevSosi"); view.reset(FloatRect(0, 0, 1000, 1000));
	bool BulGun = false; // Подобрана ли пушка
	int posX1, posY1, posX2, posY2; //вычисление
	координат char a[10] = "lox";
	Clock clock;
	float rotation = 0;
	bool isMove = false;
	int Zaderzka = 4;
	int ZaderzkaG = 9;
	int ZaderzkaZ = 0;
	float shotRed = 0;
	srand(time(0));
	int Value = 20;
	float respavn = 0;

	vector<float> tempX; //координаты курсора в
	пулю vector<float> tempY;
	vector<int> pX; //координаты игрока в пули
```

```
      vector<int> pY;

      Player p("Hero.png", 15, 0, 61, 96, 500, 600);

    vector<Entity>VecEntity;

    for(int i = 0; i < 12; i++)
    {
      VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100, 1700, 2300));
    }


      Font font;
      font.loadFromFile("FontText.otf");

      Text text;
      text.setFont(font);
      text.setString(a);
      text.setCharacterSize(32);
      text.setFillColor(Color::Black);
      text.setStyle(Text::Bold);

      Text text1;
      text1.setFont(font);
      text1.setString(a);
      text1.setCharacterSize(20);
      text1.setFillColor(Color::Black);
      text1.setStyle(Text::Bold);
//=======================================
      Image map_image;
      map_image.loadFromFile("images/itemsAll.png");
      map_image.createMaskFromColor(Color(255, 255,
      255)); Texture map;
      map.loadFromImage(map_image);
      Sprite s_map;
      s_map.setTexture(map);

      SoundBuffer BufShot;
      BufShot.loadFromFile("soung/Shot.ogg");
      Sound shotS;
      shotS.setBuffer(BufShot);


      vector<Objects> VecObjects;
      vector<Objects> VecBullets;
/* VecBullets.push_back(Objects("itemsAll.png", 226, 2, 12, 7, 0, 0)); //Пуля

            VecBullets[0].sprite.setOrigin(-64, 3);*/
            VecObjects.push_back(Objects("itemsAll.png", 160, 0, 64, 32, 300,
330)); //Пушка
            VecObjects[0].sprite.setOrigin(32, 16);

//=======================================
      Music fon;
      fon.openFromFile("soung/Fon.ogg");
      fon.play();

      while(window.isOpen())
      {
            srand(time(NULL));
            float time = clock.getElapsedTime().asMicroseconds();
            clock.restart();
            time = time / 800;
```

```
            fon.setVolume(Value);


            Vector2i pixelPos = Mouse::getPosition(window);//забираем коорд
курсора
            Vector2f pos = window.mapPixelToCoords(pixelPos);//переводим их в
игровые (уходим от коорд окна)
            Event work;
            while(window.pollEvent(work))
            {
                  if(work.type == Event::Closed)
                        window.close();
            }


//ДВИЖЕНИЕ==================================
      p.control(time);

      if(p.life == true)
      {
            if(Keyboard::isKeyPressed(Keyboard::R))
            {
                  if(BulGun == false)


if(p.sprite.getGlobalBounds().intersects(VecObjects[0].sprite.getGlobalBounds(
)
))
                                          BulGun = true;

            }
            else if(Keyboard::isKeyPressed(Keyboard::Q))
                  BulGun = false;

            else if (Mouse::isButtonPressed(Mouse::Left) && BulGun == true)
            {

                  if(Zaderzka == 5)
                  {
                        shotS.play();

                        VecBullets.push_back(Objects("itemsAll.png", 226, 2,
12, 7, VecObjects[0].sprite.getPosition().x,
VecObjects[0].sprite.getPosition().y - 5)); //Пуля
                        VecBullets[VecBullets.size() -
1].sprite.setRotation(rotation);
                        isMove = true;
                        tempX.push_back(pos.x);//забираем координату нажатия
курсора X
                        tempY.push_back(pos.y);//забираем координату нажатия
курсора Y
                        Zaderzka = 0;
                  }
                  else
                        Zaderzka++;

                  if(VecBullets.size() > 30)
                  {
                        VecBullets.erase(VecBullets.begin());
                        tempX.erase(tempX.begin());
                        tempY.erase(tempY.begin());
```

```
                }
                cout << pos.x << " " << pos.y << endl;
            }
            else if(Keyboard::isKeyPressed(Keyboard::G))
            {
                if(ZaderzkaG == 10)
                {
                    VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100,
p.sprite.getPosition().x, p.sprite.getPosition().y));
                    ZaderzkaG = 0;
                }
                else
                    ZaderzkaG++;
            }
            else if(Keyboard::isKeyPressed(Keyboard::P))
            {
                if(Value < 100)
                    Value += 1;
            }
            else if(Keyboard::isKeyPressed(Keyboard::L))
            {
                if(Value > 0)
                    Value -= 1;
            }
            else if(Keyboard::isKeyPressed(Keyboard::T))
            {
                if(p.zapas > 0 && ZaderzkaZ > 10)
                {
                    if(p.health < 100)
                    {
                    p.health += 20;
                    p.zapas -= 1;
                    }
                    ZaderzkaZ = 0;
                }
                else
                    ZaderzkaZ++;
            }
        }

        /*for(int i = 0; i < int(VecBullets.size()); ++i)
        {
            if(VecBullets[i].BulletFly == true || VecBullets[i].distance >
            2) {
            VecBullets[i].strelba(time, isMove, tempX, tempY,
            VecEntity); VecBullets[i].BulletFly = false;
            }
        }*/
        for(int i = 0; i < int(VecBullets.size()); ++i)
        {
            VecBullets[i].strelba(time, isMove, tempX[i], tempY[i],
            VecEntity); if((VecBullets[i].sprite.getPosition().x > 1920 ||
VecBullets[i].sprite.getPosition().x < 0) ||
(VecBullets[i].sprite.getPosition().y > 2688
|| VecBullets[i].sprite.getPosition().y < 0))
                {
                    VecBullets.erase(VecBullets.begin() +
                    i); tempX.erase(tempX.begin() + i);
                    tempY.erase(tempY.begin() + i);
                }
        }

        respavn += time;
```

```
      if(respavn > 5000)
      {
            VecEntity.push_back(Entity("Gus.png", 0, 0, 100, 100, 1700,
            2300)); respavn = 0;
      }

            //cout << true;
////=====================================
      rotation = VecObjects[0].Rotations(BulGun, p.life, pos.x, pos.y, p.dx,
p.dy, p.sprite);
//==================================

            p.update(time);

            shotRed += 0.005 * time;
            //cout << shotRed << endl;
            for(int i = 0; i < int(VecEntity.size()); ++i)
            {
                  VecEntity[i].update(time);
                  VecEntity[i].Artificial_Intelligence(time)
                  ;
            }
            if(int(shotRed) > 3)
            {
                  for(int i = 0; i < int(VecEntity.size()); ++i)
                  {
                        VecEntity[i].sprite.setColor(Color::White);
                  }
                  //cout << "RABOTYAI TVAR";
                  shotRed = 0;
            }

            posX1 = p.sprite.getPosition().x;
            posY1 = p.sprite.getPosition().y;
            posX2  =  VecObjects[0].sprite.getPosition().x;
            posY2 = VecObjects[0].sprite.getPosition().y;



            window.setView(view);
            window.clear(Color::White);

            for(int i = 0; i < HEIGHT_MAP; i++)
                  for (int j = 0; j < WIDTH_MAP; j++)
                  {
                        if (TileMap[i][j] == '0')
s_map.setTextureRect(IntRect(0, 0, 32, 32)); //если встретили символ пробел, то
рисуем 1й квадратик
                        if (TileMap[i][j] == ' ')
s_map.setTextureRect(IntRect(32, 0, 32, 32));//если встретили символ s, то
рисуем 2й квадратик
                        if (TileMap[i][j] == 's')
s_map.setTextureRect(IntRect(64, 0, 32, 32));//если встретили символ 0, то
рисуем 3й квадратик
                        if (TileMap[i][j] == 'p')
s_map.setTextureRect(IntRect(96, 0, 32, 32));//если встретили символ 0, то
рисуем 3й квадратик
                        if (TileMap[i][j] == 'P')
s_map.setTextureRect(IntRect(224, 64, 32, 32));//если встретили символ 0, то
рисуем 3й квадратик
                        if (TileMap[i][j] == 'z')
s_map.setTextureRect(IntRect(128, 0, 32, 32));
```

```
                        if (TileMap[i][j] == 'k')
s_map.setTextureRect(IntRect(160, 64, 32, 32)); if
                        (TileMap[i][j] == 'd')
s_map.setTextureRect(IntRect(224, 32, 32, 32)); if
                        (TileMap[i][j] == 'Z')
s_map.setTextureRect(IntRect(160, 32, 32, 32)); if
                        (TileMap[i][j] == 'V')
s_map.setTextureRect(IntRect(32, 32, 32, 32)); if
                        (TileMap[i][j] == 'F')
s_map.setTextureRect(IntRect(0, 32, 32, 32));
                        if (TileMap[i][j] == 'f')
s_map.setTextureRect(IntRect(128, 64, 32, 32)); if
                        (TileMap[i][j] == 'D')
s_map.setTextureRect(IntRect(96, 32, 32, 32)); if
                        (TileMap[i][j] == 'S')
s_map.setTextureRect(IntRect(128, 32, 32, 32));


                s_map.setPosition(j * 32, i * 32);//по сути раскидывает
квадратики, превращая в карту. то есть задает каждому из них позицию. если убрать,
то вся карта нарисуется в одном квадрате 32*32 и мы увидим один квадрат

                window.draw(s_map);//рисуем квадратики на

                экран }

            //нижний правый угол
            ostringstream playerScore;
            ostringstream playerHealth;
            playerScore << p.PlayerPoint;
            playerHealth << p.health << " " << p.zapas;
            text.setString("Lox: " + playerScore.str() + "\nHeatlh: " +
playerHealth.str());
            text.setPosition(view.getCenter().x + 300, view.getCenter().y +
400);

            //нижний левый угол
            ostringstream x1;
            x1 << posX1 << " " << posY1 << " " << posX2 << " " << posY2;
            text1.setString(x1.str());
            text1.setPosition(view.getCenter().x - 300, view.getCenter().y +
400);

            window.draw(p.sprite);
            window.draw(text);
            window.draw(text1);

            for (int i = 0; i < int(VecObjects.size()); ++i)
                        {
                        Objects Obj = VecObjects[i];
                        window.draw(Obj.sprite);
                        }
            for (int i = 0; i < int(VecBullets.size()); ++i)
                        {
                        Objects Bull = VecBullets[i];
                        window.draw(Bull.sprite);
                        }

            for (int i = 0; i < int(VecEntity.size()); ++i)
                        {
                        Entity Gus = VecEntity[i];
                        window.draw(Gus.sprite);
                        }
```

```
            window.display();
      }


      return 0;
}
```

10.2.2.2. ClassPlayer.hpp

```
#include "map.hpp"
#include "viev.hpp"

using namespace std;
using namespace sf;

class Player
{
public:
      float x, y, w, h, dx, dy, speed = 0, fx,
      fy; int PlayerPoint = 0;
      float CurrentFrame = 0;
      int dir = 1, health = 100;
      bool life = true, isPlayer = true;
      int BulGun = 1; // Подобрана ли пушка
      int zapas = 0;
      String File;
      Image image;
      Texture texture;
      Sprite sprite;

      Player(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);
      texture.loadFromImage(image);
      sprite.setTexture(texture);
      sprite.setTextureRect(IntRect(x, y, w, h));

      }

      void control(float time)
      {
            changeView();
            if(life == true)
            {
                  if(Keyboard::isKeyPressed(Keyboard::Left))
                  {
                        dir = 1; speed = 0.2;
                        CurrentFrame += 0.005 * time;
                        if(CurrentFrame > 3)
                              CurrentFrame = 0;

                        sprite.setTextureRect(IntRect(16 + (int(CurrentFrame) *
100), 0, 61, 96));
                        getPlayerCoordinateForView(sprite.getPosition().x,
sprite.getPosition().y);
                  }
                  else if(Keyboard::isKeyPressed(Keyboard::Right))
```

```
                {
                        dir = 0; speed = 0.2;
                        CurrentFrame += 0.005 * time;
                        if(CurrentFrame > 3)
                              CurrentFrame = 0;

                        sprite.setTextureRect(IntRect(77 + (int(CurrentFrame) *
100),0, -61,96));
                        getPlayerCoordinateForView(sprite.getPosition().x,
sprite.getPosition().y);
                }
                else if(Keyboard::isKeyPressed(Keyboard::Up))
                {
                        dir = 3; speed = 0.2;
                        CurrentFrame += 0.005 * time;
                        if(CurrentFrame > 2)
                              CurrentFrame = 0;

                        sprite.setTextureRect(IntRect(222, 101 +
(int(CurrentFrame) * 100), 51, 98));
                        getPlayerCoordinateForView(sprite.getPosition().x,
sprite.getPosition().y);
                }
                else if(Keyboard::isKeyPressed(Keyboard::Down))
                {
                        dir = 2; speed = 0.2;
                        CurrentFrame += 0.005 * time;
                        if(CurrentFrame > 2)
                              CurrentFrame = 0;

                        sprite.setTextureRect(IntRect(222, 101 +
(int(CurrentFrame) * 100), 51, 98));
                        getPlayerCoordinateForView(sprite.getPosition().x,
sprite.getPosition().y);
                        }
                else
                {
                        sprite.setTextureRect(IntRect(119,100,57,92));
                }
          }
          else if(Keyboard::isKeyPressed(Keyboard::W)) //Тыкать если сдох
          {
                life = true;
                health = 100;//здоровеь снова 100
          }
          else
                sprite.setTextureRect(IntRect(0,200,100,100));
    }


    void update(float time)
    {
    switch(dir)
          {
          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;
          }
          dx += fx * time;
          dy += fy * time;

          speed = 0;
```

```
            sprite.setPosition(dx, dy);
            interactionWithMap();

            if(health <= 0)
                  life = false;
      }


      void interactionWithMap()
      {
            for(int i = dy / 32; i < (dy + h) / 32; i++)
            for(int j = dx / 32; j < (dx + w) / 32; j++)
            {
                  if(TileMap[i][j] == '0' || TileMap[i][j] == 'd' ||
TileMap[i][j] == 'Z')
                  {
                        if (fy > 0)//если мы шли вниз,
                                          dy = 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)
                                          dy = i * 32 + 32;//аналогично с
   ходьбой вверх. dy<0, значит мы идем вверх (вспоминаем координаты паинта)
                        if (fx > 0)
                                          dx = j * 32 - w;//если идем вправо,
то координата X равна стена (символ 0) минус ширина
                        персонажа if (fx < 0)
                                          dx = j * 32 + 32;//аналогично идем
влево
                  }

                  if(isPlayer == true)
                  {
                        if(TileMap[i][j] == 'p') //Взятие объекта
                        PY {
                              PlayerPoint++;
                              health -= 40;
                              TileMap[i][j] = ' ';
                        }
                        if(TileMap[i][j] == 'P') //Взятие объекта PY
                        {
                              PlayerPoint++;
                              health -= 40;
                              TileMap[i][j] = 'k';
                        }
                        if(TileMap[i][j] == 'z') //Взятие объекта zelie
                        {
                              zapas += 1;
                              TileMap[i][j] = ' ';
                              randPy('z');
                        }
                  }
            }
      }
};

class Entity : public Player
{
public:
      bool dv = false; // что б много питонов не спавнили
      Entity(String F, float X, float Y, float W, float H, float dX, float
dY) : Player(F, X, Y, W, H, dX, dY)
      {
```

```
	isPlayer = false;
	}

void Artificial_Intelligence(float time)
	{
		int randGus = 1 + rand() % 10;
			if (randGus == 1 || randGus == 5)
			{
				dir = 1; speed = 0.2; dv = true;
				CurrentFrame += 0.005 * time;
				if(CurrentFrame > 3)
					CurrentFrame = 0;

				sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) *
100), 0, 100, 100));
			}
			if (randGus == 2 || randGus == 6)
			{
				dir = 0; speed = 0.2; dv = true;
				CurrentFrame += 0.005 * time;
				if(CurrentFrame > 3)
					CurrentFrame = 0;

				sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) *
100), 0, 100, 100));
			}
			if (randGus == 3 || randGus == 7)
			{
				dir = 3; speed = 0.2; dv = true;
				CurrentFrame += 0.005 * time;
				if(CurrentFrame > 3)
					CurrentFrame = 0;

				sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) *
100), 0, 100, 100));
			}
			if (randGus == 4 || randGus == 8)
			{
				dir = 2; speed = 0.2; dv = true;
				CurrentFrame += 0.005 * time;
				if(CurrentFrame > 3)
					CurrentFrame = 0;

				sprite.setTextureRect(IntRect(0 + (int(CurrentFrame) *
100), 0, 100, 100));
			}
			if (randGus == 9 && TileMap[(int)sprite.getPosition().y /
32] [(int)sprite.getPosition().x / 32] != '0' && dv == true)
			{
				if(TileMap[(int)sprite.getPosition().y / 32]
[(int)sprite.getPosition().x / 32] == ' ')
					TileMap[(int)sprite.getPosition().y /
32] [(int)sprite.getPosition().x / 32] = 'p';
				if(TileMap[(int)sprite.getPosition().y / 32]
[(int)sprite.getPosition().x / 32] == 'k')
					TileMap[(int)sprite.getPosition().y /
32] [(int)sprite.getPosition().x / 32] = 'P';
				 dv = false;
			}
	}

};
```

```
class Objects : public Player
{
public:
	//float distance = 0;
	float rotation = 0;
	float VecX = 0, VecY = 0;
	bool shot = false;

	Objects(String F, float X, float Y, float W, float H, float dX, float dY)
: Player(F, X, Y, W, H, dX, dY)
	{
	sprite.setPosition(dx, dy);
	isPlayer = false;
	sprite.setOrigin(w / 2, h / 2);
	}

	void strelba(float time , bool isMove, float posX, float posY,
vector<Entity> &Entit)
	{
		if(isMove)
		{
			//    distance = sqrt((posX - sprite.getPosition().x -
50)*(posX - sprite.getPosition().x  - 50) + (posY -
sprite.getPosition().y)*(posY - sprite.getPosition().y));//считаем
дистанцию (длину от точки А до точки Б). формула длины вектора
			for(int i = 0; i < int(Entit.size()) && shot == false; i++)
				{


if(sprite.getGlobalBounds().intersects(Entit[i].sprite.getGlobalBounds()))
					{
						Entit[i].health -= 40;
						shot = true;
						//cout << Entit[i].health << endl;
						Entit[i].sprite.setColor(Color::Red);
						if(Entit[i].health < 0)
						{
						Entit.erase(Entit.begin() + i);
						}
					}
				};
				//cout <<  << endl;
			if (shot != true)
			{//этим условием убираем дергание во время конечной позиции
спрайта
				if(TileMap[(int)sprite.getPosition().y / 32]
[(int)sprite.getPosition().x / 32] != '0' &&
TileMap[(int)sprite.getPosition().y / 32][(int)sprite.getPosition().x / 32]
!= 'd' && TileMap[(int)sprite.getPosition().y /
32][(int)sprite.getPosition().x / 32] != 'Z')
				{
					float lineX = dx - posX;
					float lineY = dy - posY;
					float line = sqrt((lineX * lineX) + (lineY *
lineY));
					float Xx = (lineX / (line));
					float Yy = (lineY / (line));
					//cout << (Xx * Xx) +(Yy * Yy) << endl;
					sprite.move(-(2 * Xx) * time, -(2 * Yy) * time);
				}
			}
```

```
                else
                        isMove = false;
            }
      }
      //

      int Rotations(bool Gun, bool life, float pos_x, float pos_y, float p_dx,
float p_dy, Sprite A)
      {
            if(Gun == true && life == true)
            {
                  VecX = pos_x - p_dx;//вектор , колинеарный прямой, которая
пересекает спрайт и курсор
                  VecY = pos_y - p_dy;//он же, координата y
                  rotation = (atan2(VecY, VecX)) * 180 /
3.14159265;//получаем угол в радианах и переводим его в градусы

                  sprite.setRotation(rotation);//поворачиваем спрайт на эти
градусы

                  if(rotation >= -90 && rotation <= 90)
                  {
                        sprite.setTextureRect(IntRect(160, 0, 64, 32));
                        sprite.setPosition(A.getPosition().x + 30,
A.getPosition().y + 45);
                  }
                  else
                  {
                        sprite.setTextureRect(IntRect(160, 32, 64, -
                        32)); sprite.setPosition(A.getPosition().x + 20,
A.getPosition().y + 45);
                  }
            }
            return rotation;
      }
};
```

10.2.2.3. Viev.hpp

```
using namespace sf;

View view;

void getPlayerCoordinateForView(float x, float y)
{
      float tempX = x; float tempY = y;

      if (x < 400) tempX = 400;//убираем из вида левую сторону
      if (x > 1320) tempX = 1320;//убираем из вида левую
      сторону if (y < 400) tempY = 400;//верхнюю сторону
      if (y > 2088) tempY = 2088;//нижнюю сторону

      view.setCenter(tempX + 100, tempY + 100);

}

void changeView()
{
      if(Keyboard::isKeyPressed(Keyboard::U))// отдаление
            view.zoom(1.005);
      if(Keyboard::isKeyPressed(Keyboard::J))// приближение
            view.zoom(0.9995);
      if (Keyboard::isKeyPressed(Keyboard::I))
```

```
            view.setSize(1000, 1000);//устанавливает размер камеры (наш
исходный)
}
```

10.2.2.4. Map.hpp

```
using namespace sf;

const int HEIGHT_MAP = 84;//размер карты высота

const int WIDTH_MAP = 60;//размер карты ширина


String TileMap[HEIGHT_MAP] = {
"0000000000000000000000000000000000000000000000000ddddddddddddd",
"0                                                 ddddddddddddd",
"0                                                 ddkkkkkkkkdd",
"0                                                 ddkkkkkkkkdd",
"0                         z                       ddkkkkkkkkdd",
"0                                                 ddkkkkkkkkdd",
"0   dZZkkkkkkddd                     D            ddkkkkkkkkdd",
"0   dddkkkkkkddd                                  ddkkkkkkkkdd",
"0   ddkkkkkkkkZZ                                  ZZkkkkkkkkdd",
"0   ddkkkkkkkkddS                                 ddkkkkkkkkdd",
"0   ddkkkkkkkkddS                                 ddkkkkkkkkdd",
"0   ddkkkkkkkkdd                                  ddkkkkkkkkdd",
"0   ddkkkkkkkkdd                     ddddddZZddddddkkkkkkkkdd",
"0   ddkkkkkkkkdd                     dddZZZZddddddddkkkkkkkkdd",
"0   ddkkkkkkkkdd                     ddkkkkkkkkkkkkkkkkkkkkdd",
"0   ddkkkkkkkkdd                     kkkkkkkkkkkkkkkkkkkkkkdd",
"0   dZZZZZZddddd                     kkkkkkkkkkkkkkkkkkkkkkdd",
"0   ddddddZZZZZd                     kkkkkkkkkkkkkkkkkkkkkkdd",
"0                                z    kkkkkkkkkkkkkkkkkkkkkkZZ",
"0                                     kkkkkkkkkkkkkkkkkkkkkkZZ",
"0                                     kkkkkkkkkkkkkkkkkkkkkkdZ",
"0                                     ddkkkkkkkkkkkkkkkkkkkkdZ",
"0                                     ddddddddddddddddkkkkkkkdZ",
"0                                     ddddddddZZZddddkkkkkkkkdd",
"0                                                 Sddkkkkkkkkdd",
"0                                                  ddkkkkkkkkdZ",
"0                                                  ddkkkkkkkkdd",
"0                         D                        ddkkkkkkkkdd",
"0                                                  Zdkkkkkkkkdd",
"0                                        D         ddkkkkkkkkdd",
"0                                                  ddkkkkkkkkdd",
"0                                                  ddkkkkkkkkdd",
"0                                                  ddkkkkkPkkdZ",
"0                                   ddddddddddddddddddkkkkkkkkdd",
"0                    D              ddddddddddddddddddkkkkkkkkdd",
"0                                                  ddkkkkkkkkdd",
"0ffffffffffffffffffffV                             ddkkkkkkkkZd",
"0                     F                            ddkkkkkkkkdd",
"0fffffffffffffffV     F                            ddkkkkkkkkdd",
"0                  F  F  z                         ddkkkkkkkkdd",
"0    D       D     F  F                            ddkkkkkkkkdd",
"0                  F  F                            ddkkkkkkkkdd",
"0  dddkkkkkkdddS   F  F              D        z    ddkkkkkkkkdd",
"0  dddkkkkkkddd    F  F                           ZZkkkkkkkkdd",
"0  ddkkkkkkkkdd    F  F                           ddkkkkkkkkdd",
"0  ddkkkkkkkkdd    F  F                           ZZkkkkkkkkdd",
"0  ddkkkkkkkkdd    F  F                           ddkkkkkkkkdd",
"0  ddkkkkkkkkdd    F  F                 SS        ddkkkkkkkkdd",
"dddddkkkkkkkkdd    F  F              ddddddddddddddddkkkkkkkkdd",
```

```
"dddddkkkkkkkkdd   F   F            ZZZZZZZddddddddkkkkkkkkdd",
"ZZkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdZ   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkZZ   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkZZ   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkZZ   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdZ   F   F           SZdkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F          SSddkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkkkkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F            ddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdd   F   F           Sddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkddS                 SSddkkkkkkkkkkkddkkkkkkkkdd",
"ddkkkkkkkkkkkdddddddZZZZZZddddddddddddddddddddddddkkkkkkkkdd"
,
"ddkkkkkkkkkkkddddddddddZZdddddddddddddddddddddddddkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"Zdkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"Zdkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"Zdkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"dZkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"dZkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"dZkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ZZkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"ddkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdd"
,
"dddddddddddddddddddddddddddddddZZZZZZZZZZZZZZZZZdddddddddddd"
,
"dddddddddddddddddddZZZZZZZZZZZZZZddddddddddddddddddddddddddd"
};

void randPy(char a)
{
	int randX, randY;
	srand(time(0));
	int countPy = 1;

	while(countPy > 0)
	{
		randX = 1 + rand() % (HEIGHT_MAP - 1);
		randY = 1 + rand() % (WIDTH_MAP - 1);
		if(TileMap[randX][randY] == ' ')
		{
```

```
                TileMap[randX][randY] = a;
                countPy--;
            }
        }
}
```

10.2.3. Листинг 6 – 8. Второй режим

10.2.3.1. TanksGame.hpp

```
#include "resourse/TanksClass.hpp"
```

```
using namespace std;
using namespace sf;

int tanks()
{
	RenderWindow window(VideoMode(1024, 1024),
	"Tanks"); Clock clock;
	int vie2 = 3, vie22 = 3;
	float timeresp = 0;
	int randResp = 0;
	int Score = 0, ScoreMax = 0, Destroyed = 0, DestroyedMax = 0;
	int Score2 = 0, ScoreMax2 = 0, Destroyed2 = 0, DestroyedMax2 =
	0; Player2 player("Tanks.png", 240, 20 , 40, 40, 942, 942);
	Player2 player2("Tanks.png", 240, 20 , 40, 40, 942, 902);
	//player.sprite.setOrigin(player.h / 2, player.w / 2);
	Player2 map("itemsAll.png", 0, 0, 32, 32, 0, 0);

	objc pul("itemsAll.png", 160, 1, 9, 17, -15, -15);
	objc pul2("itemsAll.png", 160, 1, 9, 17, -15, -15);
	vector<Entity2>VecEntity;
	vector<objc>VecPul;
	vector<int> vie3;

	Font font;
	font.loadFromFile("resourse/FontText.otf");

	Text text;
	text.setFont(font);
	text.setCharacterSize(28);
	text.setFillColor(Color::White);
	text.setStyle(Text::Bold);

	Text text2;
	text2.setFont(font);
	text2.setCharacterSize(28);
	text2.setFillColor(Color::White);
	text2.setStyle(Text::Bold);

	for(int i = 0; i < 4; i++)
	{
		VecEntity.push_back(Entity2("Tanks.png", 190, 20, 40, 40, 32,
		32)); VecPul.push_back(objc("itemsAll.png", 160, 1, 9, 17, -15, -
		15)); vie3.push_back(0);
	}

	while(window.isOpen())
	{
		window.clear(Color::White);
		srand(time(NULL));
		float time = clock.getElapsedTime().asMicroseconds();
		clock.restart();
		time = time / 800;

		timeresp += time;
		if(timeresp >= 5000)
		{
			randResp = 1 + rand() % 3;
			if(randResp == 1)
				VecEntity.push_back(Entity2("Tanks.png", 190, 20, 40,
40, 32, 32));
```

```
                  if(randResp == 2)
                        VecEntity.push_back(Entity2("Tanks.png", 190, 20, 40,
40, 942, 32));
                  if(randResp == 3)
                        VecEntity.push_back(Entity2("Tanks.png", 190, 20, 40,
40, 32, 942));

                  VecPul.push_back(objc("itemsAll.png", 160, 1, 9, 17, -15, -
15));
                  vie3.push_back(0);
                  timeresp = 0;
            }

            Event game;
            while(window.pollEvent(game))
            {
                  if(game.type == Event::Closed)
                        window.close();
            }

            if (player.life == true)
            {
                  if(Keyboard::isKeyPressed(Keyboard::Space))
                  {
                        if(pul.isMove == false)
                        {
                              pul.sprite.setPosition(player.dx + 20, player.dy
+ 20);
                              pul.isMove = true;
                              vie2 = player.vie;
                        }
                  }
            }

            if (player2.life == true)
            {
                  if(Keyboard::isKeyPressed(Keyboard::C))
                  {
                        if(pul2.isMove == false)
                        {
                              pul2.sprite.setPosition(player2.dx + 20,
player2.dy + 20);
                              pul2.isMove = true;
                              vie22 = player2.vie;
                        }
                  }
            }

            pul.strelba(time, vie2);
            pul2.strelba(time, vie22);

            player.control(time);
            player.update(time);
            player2.control2(time);
            player2.update(time);

      for(int i = 0; i < int(VecEntity.size()); ++i)
            {
                  VecEntity[i].Artificial_Intelligence(time)
                  ; VecEntity[i].update(time);
                  VecPul[i].strelba(time, vie3[i]);
                  if(VecPul[i].isMove == false)
```

```
                  {
                        vie3[i] = VecEntity[i].vie;
                        VecPul[i].sprite.setPosition(VecEntity[i].dx + 20,
VecEntity[i].dy + 20);
                        VecPul[i].isMove = true;
                  }



if(pul.sprite.getGlobalBounds().intersects(VecEntity[i].sprite.getGlobalBounds( )))

                  {
                        VecEntity.erase(VecEntity.begin() + i);
                        VecPul.erase(VecPul.begin() + i);
                        pul.isMove = false;
                        pul.sprite.setPosition(-15, -15);
                        Score += 20;
                        Destroyed++;
                  }


if(pul2.sprite.getGlobalBounds().intersects(VecEntity[i].sprite.getGlobalBound
s ()))
                  {
                        VecEntity.erase(VecEntity.begin() + i);
                        VecPul.erase(VecPul.begin() + i);
                        pul2.isMove = false;
                        pul2.sprite.setPosition(-15, -15);
                        Score2 += 20;
                        Destroyed2++;
                  }



if(VecPul[i].sprite.getGlobalBounds().intersects(player.sprite.getGlobalBounds( )))

                  {
                        player.dx = 952;
                        player.dy = 952;
                        Score = 0;
                        Destroyed = 0;
                  }


if(VecPul[i].sprite.getGlobalBounds().intersects(player2.sprite.getGlobalBound
s ()))
                  {
                        player2.dx = 952;
                        player2.dy = 952;
                        Score2 = 0;
                        Destroyed2 = 0;
                  }
            }

            if(ScoreMax < Score)
                  ScoreMax = Score;
            if(DestroyedMax < Destroyed)
                  DestroyedMax = Destroyed;

            if(ScoreMax2 < Score2)
                  ScoreMax2 = Score2;
```

```
            if(DestroyedMax2 < Destroyed2)
                  DestroyedMax2 = Destroyed2;

            for(int i = 0; i < HEIGHT_MAP2; i++)
                  for(int j = 0; j < WIDHT_MAP2; j++)
                  {
                              if(TileMap2[i][j] == '0')
   map.sprite.setTextureRect(IntRect(0, 0, 32, 32));
                              if(TileMap2[i][j] == ' ')
  map.sprite.setTextureRect(IntRect(32, 0, 32, 32));

                        map.sprite.setPosition(j * 32, i * 32);

                        window.draw(map.sprite);

                   }
            ostringstream Scored;
            ostringstream Scored2;
            Scored << "Score: " << Score << "      " << "Destroyed: " <<
Destroyed << "      ";
            Scored << "Record Score: " << ScoreMax << "     " << "Record
Destroyed: " << DestroyedMax;
            Scored2 << "Score2: " << Score2 << "     " << "Destroyed2: " <<
Destroyed2 << "      ";
            Scored2 << "Record Score2: " << ScoreMax2 << "     " << "Record
Destroyed2: " << DestroyedMax2;


            text.setString(Scored.str());
            text.setPosition(10, -2);
            text2.setString(Scored2.str());
            text2.setPosition(10, 992);



            window.draw(player.sprite);
            window.draw(player2.sprite);
            window.draw(pul.sprite);
            window.draw(pul2.sprite);
            window.draw(text);
            window.draw(text2);
            for(int i = 0; i < int(VecEntity.size()); ++i)
                  {
                        Entity2 TankEntity = VecEntity[i];
                        window.draw(TankEntity.sprite);
                  }

            for(int i = 0; i < int(VecPul.size()); ++i)
            {
                  objc PulEntity = VecPul[i];
                  window.draw(PulEntity.sprite);
            }

            window.display();
      }

      return 0;
}
```

10.2.3.2. TanksClass.hpp

```
#include "map2.hpp"

using namespace std;
using namespace sf;

class Player
{
public:
	float x, y, w, h, dx, dy, fx, fy;
	float speed = 0.0;
	int dir = 0, vie = 3, health = 100;
	bool life = true;
	String File;
	Image image;
	Texture texture;
	Sprite sprite;

	Player(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);
		texture.loadFromImage(image);
		sprite.setTexture(texture);
		sprite.setTextureRect(IntRect(x,y,w,h));
		sprite.setPosition(dx, dy);
	}

	void control(float time)
	{
		if(life == true)
		{
			if(Keyboard::isKeyPressed(Keyboard::Left))
			{
				dir = 1; speed = 0.2;
				sprite.setTexture(texture);
				sprite.setTextureRect(IntRect(280, 70, -40, 40));
				vie = 1;
			}
			if(Keyboard::isKeyPressed(Keyboard::Right))
			{
				dir = 0; speed = 0.2;
				sprite.setTexture(texture);
				sprite.setTextureRect(IntRect(240, 70, 40, 40));
				vie = 2;
			}
			if(Keyboard::isKeyPressed(Keyboard::Up))
			{
				dir = 3; speed = 0.2;
				sprite.setTexture(texture);
				sprite.setTextureRect(IntRect(240, 20, 40, 40));
				vie = 3;
			}
			if(Keyboard::isKeyPressed(Keyboard::Down))
			{
				dir = 2; speed = 0.2;
				sprite.setTexture(texture);
				sprite.setTextureRect(IntRect(280, 60, -40, -40));
				vie = 4;
			}
```

```
		}
	}

	void update(float time)
	{
		switch(dir)
		{
			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;
		}
		dx += fx * time;
		dy += fy * time;

		speed = 0;
		sprite.setPosition(dx, dy);
		interactionWithMap();
	}

	void interactionWithMap()
	{
		for(int i = dy / 32; i < (dy + h) / 32; i++)
		for(int j = dx / 32; j < (dx + w) / 32; j++)
		{
			if(TileMap[i][j] == '0')
			{
				if (fy > 0)//если мы шли вниз,
					dy = 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)
					dy = i * 32 + 32;//аналогично с
   ходьбой вверх. dy<0, значит мы идем вверх (вспоминаем координаты паинта)
				if (fx > 0)
					dx = j * 32 - w;//если идем вправо,
то координата X равна стена (символ 0) минус ширина
				персонажа if (fx < 0)
					dx = j * 32 + 32;//аналогично идем
влево
			}
		}
	}


};

class Entity : public Player
{
public:
	Entity(String F, float X, float Y, float W, float H, float dX, float
dY) : Player(F, X, Y, W, H, dX, dY)
	{

	}

	void Artificial_Intelligence(float time)
	{
		int randGus = 1 + rand() % 10;
			if (randGus == 1 || randGus == 5)
			{
				dir = 1; speed = 0.2; vie = 1;
				sprite.setTextureRect(IntRect(230, 70, -40, 40));
```

```
                  }
                  if (randGus == 2 || randGus == 6)
                  {
                        dir = 0; speed = 0.2; vie = 2;

                        sprite.setTextureRect(IntRect(190, 70, 40, 40));

                  }
                  if (randGus == 3 || randGus == 7)
                  {
                        dir = 3; speed = 0.2; vie = 3;
                        sprite.setTextureRect(IntRect(190, 20, 40, 40));
                  }
                  if (randGus == 4 || randGus == 8)
                  {
                        dir = 2; speed = 0.2; vie = 4;
                        sprite.setTextureRect(IntRect(230, 60, -40, -40));
                  }
      }
};

class objc : public Player
{
public:

      bool isMove = true;
      objc(String F, float X, float Y, float W, float H, float dX, float dY) :
Player(F, X, Y, W, H, dX, dY)
      {

      }

      void strelba(float time, int vie)
      {
            if(TileMap[(int)sprite.getPosition().y / 32]
[(int)sprite.getPosition().x / 32] != '0' && isMove == true)
            {
                  switch(vie)
                  {
                        case 1: sprite.move(-0.5 * time, 0);
sprite.setRotation(270); break;
                        case 2: sprite.move(0.5 * time, 0);
sprite.setRotation(90); break;
                        case 3: sprite.move(0, -0.5 * time);
sprite.setRotation(0); break;
                        case 4: sprite.move(0, 0.5 * time);
sprite.setRotation(180); break;
                        default: isMove = false;
                  }
            }
            else
            {
                  isMove = false;
                  sprite.move(0, 0);
                  sprite.setPosition(-15, -15);
            }
      }
};
```

10.2.3.3. map2.hpp

```
using namespace sf;
```

```
const int HEIGHT_MAP2 = 32;
const int WIDHT_MAP2 = 32;

String TileMap2[HEIGHT_MAP2] = {
      "00000000000000000000000000000000"
      ,
      "0  0                            0",
      "0  0                            0",
      "0  0         000000000000   00000",
      "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             00     0    000    0",
      "0                    0    000    0",
      "0                    0    000    0",
      "0                    0           0",
      "0                    0           0",
      "0                    0           0",
      "0                    0           0",
      "0                    0           0",
      "0    000000000000000          0  0",
      "0                             0  0",
      "0                             0  0",
      "0   00000000000000            0  0",
      "0                             0  0",
      "0                             0  0",
      "0000                          0  0",
      "0                             0  0",
      "0                             0  0",
      "00000000000000000000000000000000"
};
```

10.2.4. Листинг 9. Третий режим

10.2.4.1. minigame1.hpp

```
using namespace std;
using namespace sf;

int KrestNol()
{
      RenderWindow window(VideoMode(600, 700), "GameKO");

      int masMain[3][3];
      int a = 4, b = 4, xod = 0, KrNO, gg = 0, kk =
      0; bool game = true;
      char TEXT[6] = "Win: ";

      for(int i = 0; i < 3; i++)
            for(int j = 0; j < 3; j++)
                  masMain[i][j] = 2;

      Image Pole;
      Pole.loadFromFile("images/MiniGame/Pole.png");
      Texture PoleT;
      PoleT.loadFromImage(Pole);
```

```
	Sprite PoleS;
	PoleS.setTexture(PoleT);

	Font font;
	font.loadFromFile("FontText.otf");

	Text text;
	text.setFont(font);
	text.setString(TEXT);
	text.setCharacterSize(40);
	text.setFillColor(Color::Red);
	text.setPosition(50, 620);

	ostringstream x1;

	while(window.isOpen())
	{
		Vector2i Pos = Mouse::getPosition(window);//забираем коорд
		курсора Vector2f pixelPos = window.mapPixelToCoords(Pos);
		//смотрим на координату X позиции курсора в консоли (она не будет
больше ширины окна)
		Event work;
		while(window.pollEvent(work))
		{
			if(work.type == Event::Closed ||
Keyboard::isKeyPressed(Keyboard::X))
				window.close();
			else if(Keyboard::isKeyPressed(Keyboard::Z))
			{
				window.close();
				KrestNol();
			}
		}
			if(game)
			{
				a = pixelPos.x / 200;
				b = pixelPos.y / 200;
				if (Mouse::isButtonPressed(Mouse::Left) && kk != 9 &&
masMain[b][a] == 2)//если нажата клавиша мыши
				{
					//cout << pixelPos.x << " " << pixelPos.y << "\
n";
					if(pixelPos.x > 0 && pixelPos.y > 0)
					{


						KrNO = xod % 2;
						xod++;
						//cout << a << " " << b << "\n";
						//cout << "isClicked!\n";
						masMain[b][a] = KrNO;
						kk++;
					}
					for(int i = 0; i < 3; i++)
					{
						if(masMain[i][0] == masMain[i][1] &&
masMain[i][1] == masMain[i][2] && masMain[i][0] != 2)
						{
							//cout << "\nWIN\n";
							game = false;
						}
```

```
                                    if(masMain[0][i] == masMain[1][i] &&
masMain[1][i] == masMain[2][i] && masMain[0][i] != 2)
                                    {
                                          //cout << "\nWIN\n";
                                                game = false;
                                    }
                              }
                              if(masMain[0][0] == masMain[1][1] && masMain[1]
[1] == masMain[2][2] && masMain[0][0] != 2)
                                    {
                                          //cout << "\nWIN\n";
                                          game = false;
                                    }
                              if(masMain[0][2] == masMain[1][1] && masMain[1]
[1] == masMain[2][0] && masMain[0][2] != 2)
                                    {
                                          //cout << "\nWIN\n";
                                          game = false;
                                    }

                        }
                        else if(gg == 0 && kk == 9)
                        {
                              x1 << "draw";
                              text.setString(TEXT + x1.str());
                              text.setPosition(50, 620);
                              gg = 1;
                        }
                  }
                  else if(gg == 0)
                  {
                        if(KrNO == 0)
                              x1 << "krestik";
                        else
                              x1 << "nolik";
                        text.setString(TEXT + x1.str());
                        text.setPosition(50, 620);
                        gg = 1;

                        //window.draw(PoleS);

                  }



            for(int i = 0; i < 3; i++)
                  for(int j = 0; j < 3; j++)
                  {
                  if(masMain[i][j] == 1)
                        PoleS.setTextureRect(IntRect(0,0,200,200));
                  if(masMain[i][j] == 0)
                        PoleS.setTextureRect(IntRect(200,0,200,200));
                  if(masMain[i][j] == 2)
                        PoleS.setTextureRect(IntRect(400,0,200,200));

                  PoleS.setPosition(j * 200, i * 200);
                  window.draw(PoleS);
                  }


            window.draw(text);
            window.display();
```

```
    }
    return 0;
}
```

10.2.5. Листинг 10 - 11. Меню

10.2.5.1. Menu.hpp

```
#include "menuClass.hpp"

using namespace std;
using namespace sf;

int launcher()
{
    RenderWindow window(VideoMode(800, 600), "MenuBobnevGame");

    float CurrentFrame = 0;
    int Value = 100;
    bool isOpen = false;
    Clock clock;

    Music music;
    music.openFromFile("soung/MusMenu.ogg");
    music.play();

    vector<Button>VecButton;

    VecButton.push_back(Button("Menu.png", 0, 0, 800, 600, 0 ,0)); //Фон
    VecButton.push_back(Button("ButtonsMenu.png", 0, 0, 251, 41, 500,
50)); // Играть
    VecButton.push_back(Button("ButtonsMenu.png", 0, 50, 251, 41, 500, 100));
//FAQ
    VecButton.push_back(Button("ButtonsMenu.png", 0, 100, 251, 41, 500,
150)); //Выход
    VecButton.push_back(Button("ButtonsMenu.png", 270, 0, 40, 31, 20,
550)); // Громче
    VecButton.push_back(Button("ButtonsMenu.png", 310, 0, 40, 31, 60,
550)); // Тише
    VecButton.push_back(Button("ButtonsMenu.png", 0, 150, 400, 380, 1000,
1000)); // Доп инфа
    VecButton.push_back(Button("ButtonsMenu.png", 350, 0, 40, 31, 100, 550));
// Доп игра
    VecButton.push_back(Button("ButtonsMenu.png", 350, 40, 40, 32, 140,
550)); // Доп игра



    while(window.isOpen())
    {
          Vector2i pixelPos = Mouse::getPosition(window);//забираем коорд
курсора
          Vector2f pos = window.mapPixelToCoords(pixelPos);//переводим их в
игровые (уходим от коорд окна)
          for (int i = 1; i < int(VecButton.size()); ++i)
          {
                VecButton[i].ButtonSpite.setColor(Color(61, 58,
                      58)); if
(VecButton[i].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                            VecButton[i].ButtonSpite.setColor(Color::White)
                            ;
          }
```

```
        srand(time(NULL));
        float time = clock.getElapsedTime().asMicroseconds();
        clock.restart();
        time = time / 800;

        Event work;
        while(window.pollEvent(work))
        {
              if(work.type == Event::Closed)
                    window.close();
        else if(Mouse::isButtonPressed(Mouse::Left))
              {

   if(VecButton[3].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                          return 0;
                    else
if(VecButton[1].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                          return 10;
                    else
if(VecButton[4].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                    {
                          if(Value < 100)
                                Value += 10;
                    }
                    else
if(VecButton[5].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                    {
                          if(Value > 0)
                                Value -= 10;
                    }
                    else
if(VecButton[2].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                    {
                          if(isOpen == true)
                          {
                                VecButton[6].ButtonSpite.setPosition(0,
                                0); isOpen = false;
                          }
                          else
                          {
                                VecButton[6].ButtonSpite.setPosition(1000,
1000);
                                isOpen = true;
                          }
                    }
                    else
if(VecButton[7].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                          {
                                return 11;
                          }
                    else
if(VecButton[8].ButtonSpite.getGlobalBounds().contains(pos.x, pos.y))
                          {
                                return 12;
                          }
              }
        }
```

```
            music.setVolume(Value);



            CurrentFrame += 0.005 * time;
            if(CurrentFrame > 3)
                  CurrentFrame = 0;

            //cout << CurrentFrame;
            VecButton[0].ButtonSpite.setTextureRect(IntRect(0 +
   (int(CurrentFrame) * 800), 0, 800, 600));

            for (int i = 0; i < int(VecButton.size()); ++i)
                        {
                        Button But = VecButton[i];
                        window.draw(But.ButtonSpite);
                        }
            window.display();
      }

      return 0;
   }
```

10.2.5.2. menuClass.hpp

```
using namespace std;
using namespace sf;

class Button
{
public:
      float x, y, w, h, dx, dy;
      string F;
      Image ButtonImage;
      Texture ButtonTexture;
      Sprite ButtonSpite;

      Button(string File, float X, float Y, float W, float H, float dX, float
dY)
      {
            F = File;
            w = W; h = H;
            x = X; y = Y;
            dx = dX; dy = dY;
            ButtonImage.loadFromFile("images/" + File);
            ButtonTexture.loadFromImage(ButtonImage);
            ButtonSpite.setTexture(ButtonTexture);
            ButtonSpite.setTextureRect(IntRect(x, y, w, h));
            ButtonSpite.setPosition(dx, dy);
      }

};
```

10.3. Приложение 3

Дизайн – документ:

BobnebGame

Дизайн – документ

Содержание:

10.3.1.

1. Введение

Дальнейшее содержание этого документа организовано по разделам, в которых приводится следующая информация:

Раздел 2. Концепци. Данный раздел содержит основные сведение об игре – общее описание, жанр, предпосылки создания игры и ее особенности, целевая аудитория и платформа. Раздел предназначен для всех заинтересованных лиц.

Раздел 3. Описание игры с точки зрения пользователя и предлагаемых ею возможностей. Содержит данные про сюжет и принципы игры; интерфейс; графику и звуковое наполнение; состав игровых уровней. Этот раздел предназначен для всех заинтересованных лиц.

Раздел 4. Контакты. Содержит контактную информацию.

2. Концепция

2.1. Введение

Учился Леха Бобнев в колледже и по жизни очень сильно боялся уток. Однажды, по пути на учебу он заметил, что в его любимом колледже появились очень злые утки, из-за которых была сорвана учеба. И, Леха, недолго думая, стал искать способ как выгнать их из колледжа и продолжить учебу. Еще Леха любит играть в «Крестики – Нолики» и «Танчики». Но только с друзьями.

2.2. Платформа, жанр и аудитория

Игра «BobnevGame» относится к жанру 2D-шутеров без сюжета и элементами GPG, и разрабатывается только в версии для PC.

Игра ориентирована на широкую аудиторию, не содержит ограничивающего контента, минимальный возраст игрока – 8 лет. Дополнительную привлекательность игра имеет для владельцев не самой современной конфигурации PC.

Игра не использует торговые марки или другу собственность, подлежащую лицензированию.

2.3. Основные особенности игры

Ключевые особенности игры (USP):

Бесконечное количество противников. Возможность играть в свое удовольствие без привязки ко времени.

Бесконечные игровые жизни. Не придется начинать игру сначала. Невысокие требования к технике.

Две мини – игры на двоих человек.

Встроенные «Крестики - Нолики».

Встроенные «Танчики» с возможностью одиночного режима.

2.4. Описание игры

Игра делится на три режима:

Игровой процесс первого режима представляет собой поиск оружия, поиск аптечек в виде яблок, ведение боевых действий с противниками в виде Уток и в обходе установленных ловушек. Игрок без ограничений передвигается по игровой карте и занимается доступными в игре действиями. Он может осматривать территорию, собирать яблоки, подбирать и класть оружие в разных местах, отстреливать противников

или просто наблюдать за ними, собирать ловушки противника, умирать от них, пополнять свое здоровье яблоками, потраченное на ловушки уток.

Поиск оружия заключается в нахождении игроком автоматической винтовки, без которой невозможно вести боевые действия со стороны игрока.

Поиск аптечек в виде яблок является необязательной частью прохождения игры и заключается в поиске раскиданных по игровой карте яблок и подбором их в и инвентарь. Их можно применить только при потере части здоровья игроком и если игрок жив.

Боевые действия заключаются в стрельбе из автоматической винтовки по противникам, со стороны игрока, и в расстановке ловушек, называемых «Файлы Python», со стороны Уток.

Игрок, потеряв фиксированное количество здоровья становится неиграбельным до тех пор, пока не будет «воскрешен».

Игровой процесс второго режима представляет собой в управлении боевой единицей, именуемой танком, и ведение боевых действий против танков противника. За каждое уничтожение противника игрок получает очки опыта и увеличении считчика убийств. При уничтожении игрока очки опыта, счетчик убийств обнуляется, и игрок перемещается на место воскрешения и может продолжать игру без ограничений. Есть возможность просмотреть рекорды. Игра рассчитана на двух игроков с возможность игры в одиночку. Игроки по отношению друг к другу являются союзниками и не могу уничтожить друга.

Управление танком заключается в перемещении по карте, стрельбы по противникам, тактическом отступлении при ответном огне и использовании укрытий как часть тактики.

Боевые действия заключаются в попытках вести огонь по вражеским танкам, находящимся на линии огня и в их уничтожении прямым попаданием, а также в необходимости контролировать ситуацию и не попадая под ответный огонь.

Игровой процесс третьего режима заключается в игре двух игроков. Один играет «крестиками», второй – «ноликами». Игра ведется на квадратном поле 3 на 3. Игроки по очереди ставят на свободные клетки поля 3х3 знаки (один всегда крестики, другой всегда нолики). Первый,

выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает. Первый ход делает игрок, ставящий крестики.

2.5. Системные требования

Требования	**Минимальные**	**Рекомендуемые**
Операционная система	**Ubuntu 16.04 LTS и выше + установленная библиотека SFML**	
Процессор	Intel Core 3	Intel Core i5
ОЗУ	2GB	4GB
Свободное место на HDD	2GB	2GB
Видео карта	**Intel HD Graphics 630 и выше**	
ыы	Любая DirectX 5 совместимая звуковая карта	
Управление	Мышь, клавиатура	

3. Функциональная спецификация

3.1. Ход игры и сюжет

3.1.1. Первый режим

События игры разворачиваются на одной локации – Колледж Лёхи. Представляет собой осеннюю поляну с двумя зданиями. Одна маленькое, пост охраны. Второе большое – сам колледж, к которому подходит два ж/д пути, видимо, раньше там была станция электричек на которой студенты приезжали на учебу, но Утки её разрушили. Повсюду паутина и замшелые булыжники. Сложно поверить, что всего три дня назад тут во всю кипела жизнь и шла учеба.

Сюжет – Лёха, подходя в понедельник к первой паре, замечает, что с колледжем что-то не так: света в окнах нет, студенты не спешат на учебу и из помещения доносятся странные звуки «КРЯ - КРЯ». Он понимает, что это его старые враги - утки вернулись за ним. Не теряя времени, Леха бросает все свои вещи и бежит на пост охраны надеясь найти оружие против уток. Найдя винтовку, он отправляется мстить им за все, но, зайдя в помещение, он обнаруживает, что Утки научились программировать на Python… Он принес Лёхи больше проблем, чем Утки. Теперь его главные враги вместе и неизвестно кто победит в этой схватке: Лёха и пушка или Утки и Python.

3.1.2. Второй режим

События игры разворачиваются на одной локации – заброшенный город. Представляет собой окраину разрушенного бомбардировками и уличными боями города с кирпичной застройкой, четкого разделения на улицы не осталось – повсюду руины, создающие собой лабиринт.

Сюжет – отступающая дивизия союзных войск понесла тяжелые потери в борьбе за город и грузится на эшелоны для отправки в тыл на переформирование. Но вражеская танковая бригада не даёт перевести дух и организованно отступить в тыл без потери людских ресурсов и тяжелого вооружения. Командование отправляет танковый взвод в город для прикрытия отступления. Их задача продержаться до отхода потрепанных войск, дождаться подхода подкрепления, на которое мало кто надеется, и не дать танкам «Коалиции» захватить плацдарм на окраине города для нанесения удара в тыл бригады.

3.1.3. Третий режим

События игры разворачиваются на квадратной белой карте 3 на 3 разделенная черными полосами на равные квадраты.

Сюжет – отсутствует.

3.2. Физическая модель

3.2.1. Игровой мир и время

3.2.1.1. Первый режим

Действие происходит в параллельной реальности. В этой реальности главным отличием является умные и мыслящие Утки. Игровой мир ограничен кирпичной стеной по периметру. Смены дня и ночи нет.

3.2.1.2. Второй режим

Действия происходят в параллельной реальности. В этой реальности главным отличием является третья мировая война между армиями «Союзников» и «Коалиции». Смены дня и ночи нет. Игровой мир ограничен кирпичной стеной по периметру.

3.2.1.3. Третий режим

Статичное окно без особых свойств.

3.2.2. Жизненная и боевая модель

3.2.2.1. Первый режим

Каждый юнит обладают определенным уровнем здоровья (ЗДР).
Которое может уменьшаться или увеличиваться. Текущее состояние здоровья отображается только у главного героя.
Юниты не имеют возможности наносить урон (УРН), свойство наносить урон имеют только некоторые предметы.
Некоторые предметы могут пополнять здоровье. Они имеют свойство восстановления (ВСТ).

3.2.2.2. Второй режим

Юниты не обладаю уровнем здоровья. Юниты не могут наносить урон. Любое попадание фатально и ведёт к уничтожению.

3.2.2.3. Третий режим

Здоровье отсутствует. Атака отсутствует.

3.2.3. Движение

3.2.3.1. Первый режим

Юниты могут передвигаться по игровой карте. Непроходимыми участками считаются стены и границы мира. Проходимость формируется из свой объекта при загрузке игры.

3.2.3.2. Второй режим

Юниты могут передвигаться по игровой карте. Непроходимыми участками считаются стены и границы мира. Проходимость формируется из свой объекта при загрузке игры.

3.2.3.3. Третий режим

Передвижение отсутствует.

3.2.4. Формулы

3.2.4.1. Первый режим

Единицы здоровья всех юнитов нормированы и измеряются от 0 до 100. Таким образом, расчет уровня ЗДР производится по формуле:

Повреждение при атаке

ЗДР = ЗДР – УРН.

Восстановление здоровья

ЗДР = ЗДР + ВСТ.

Направление полета и скорость полета пули

Пуля летит по траектории (Xx, Yy).

Координата игрока по x – dX, по y – dY.

Координата курсора по x – posX, по y – PosY.

Вектор движения по x – LineX, по y – LineY.

Длина вектора – Line.

LineX = dx – posX, LineY = dy – posY.

Вычисление длины вектора, т.е. расстояния до цели:

Line = sqrt((lineX * lineX) + (lineY * lineY));

Вычисление Xx и Yy:

Xx = LineX / line, Yy = LineY / line.

3.2.4.2. Второй режим

Формулы расчета отсутствуют.

3.2.4.3. Третий режим

Формулы расчета отсутствуют.

3.3. Персонаж игрока

3.3.1. Первый режим

Основный персонажем является Лёха.

Лёха – студент программист колледжа. Отличник. Именно от его лица действует игрок. Он единственный кто умеет пользоваться огнестрельным оружием.

Характеристики:

Параметр	**Значение**
Здоровье (ЗДР)	100
Урон (УРН) (без винтовки)	0
Урон (УРН)(с винтовкой)	40

Персонаж Лёхи может находиться в следующих состояниях (в зависимости от состояния различаются его изображения на экране):

Неподвижен

Двигается

Мертв

3.3.2. Второй режим

Основным персонажем является Танк.

Танк – основной вид ударной мощи подразделения. Имеет зеленый камуфляж. Именно от него действует игрок. Танку достаточно одного попадания для уничтожения противника.

Персонаж Танка может находиться в следующих состояниях:

Двигается

Стреляет

3.3.3. Третий режим

Основной персонаж отсутствует.

3.4. Элементы игры

3.4.1. Первый режим

3.4.1.1. Утка

Утки – это главные враги Лёхи. Они начали враждовать ещё в детстве, когда Лёха первый раз поехал в город из своей деревне. Они очень умные и сообразительные существа. Быстро учатся и способны на примитивные ловушки. Они живут с мыслями как отомстить Лёхе за то, что он уехал из деревни.

Характеристики:

Параметр	**Значение**
Здоровье (ЗДР)	100
Урон (УРН)	0
Урон (УРН) (минами)	40

Утка может находиться в следующих состояниях:

Двигается

Ставит мину

Получает урон (изображение меняется на красный, потом возвращается в исходное)

3.4.2. Второй режим

3.4.2.1. Танк противника

Танк сил «Коалиции» представляет собой универсальную боевую единицу, способную как поддерживать пехоту, так и бороться со вражескими танками. Танку достаточно одного попадания для уничтожения противника.

Танк может находиться в следующих состояниях:

Двигается

Стреляет

3.4.3. Третий режим

3.4.3.1. Крестик

С него начинается игра. Объект первого игрока, который использует его как инструмент для победы расставляя по полю.

3.4.3.2. Нолик

Продолжает игру после хода крестика. Объект первого игрока, который использует его как инструмент для победы расставляя по полю.

3.4.3.3. Поле вывода

Пока идет игра имеет статическую надпись снизу от игрового поля «Win:». В зависимости от победы одного из игроков дописывает к своему полю либо «Krestik», либо «Nolik».

3.5. Специальные объекты

3.5.1. Первый режим

3.5.1.1. Автоматическая винтовка

Основное и единственное оружие Лёхи. Может вести стрельбу только когда подобрана. Имеет высокую скорострельность и не может быть уничтожена. Урон от одного попадания равен 40 единицам здоровья. Меняет свое положение относительно направления стрельбы.

3.5.1.2. Яблоко

Яблоко представляет собой любимую еду Лёхи. В игре используется в качестве аптечки для восстановления здоровья. Складируемый предмет. После побора Лёхой случайно ставится в другом месте на карте. Восстанавливает 20 единиц здоровья.

3.5.1.3. Файл Python

Язык программирования Python является нелюбимым языком Лёхи и Утки активно пользуются им, расставляя у него на пути и строя из них непроходимые баррикады. Исчезает после наступления на него Лёхой. Наносит урон в 40 единиц здоровья.

3.5.2. Второй режим

3.5.2.1. Снаряд

Снаряд бронебойный. Используется для уничтожения танков противника. Скорость полета маленькая, но огневая мощь огромная. Уничтожает танк от одного попадая и уничтожается сам. Так же уничтожается от соприкосновения с стенкой. Танк не может выстрелить новым снарядом до тех, пор пока предыдущий находится в полёте.

3.5.3. Третий режим

Специальные объекты отсутствуют.

3.6. «Искусственный интеллект»

«Искусственный интеллект» (далее ИИ)

3.6.1. Первый режим

Задачами ИИ является управление юнитами для передвижения и противодействию игроку.

Реализация ИИ для управления юнитами предусматривает:

Передвижение по карте в случайном направлении.

Расстановка ловушек в случайный местах.

Действия юнитов не согласованны.

3.6.2. Второй режим

Задачами ИИ является управление юнитами для передвижения и противодействию игроку.

Реализация ИИ для управления юнитами предусматривает:

Передвижение по карте в случайном направлении.

Ведение стрельбы при первой возможности

Действия юнитов не согласованны.

3.6.3. Третий режим

ИИ отсутствует.

3.7. Многопользовательский режим

3.7.1. Первый режим

Игра не предполагает использование многопользовательского режима

3.7.2. Второй режим

Многопользовательский режим представляет собой использовании двумя игроками разных клавиш для управления разными игровыми Юнитами.

Многопользовательский режим доступен сразу, без специальных настроек.

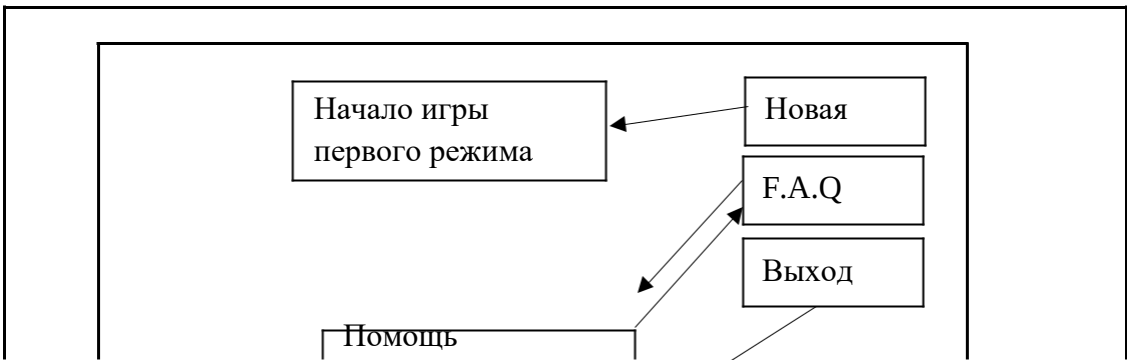
3.7.3. Третий режим

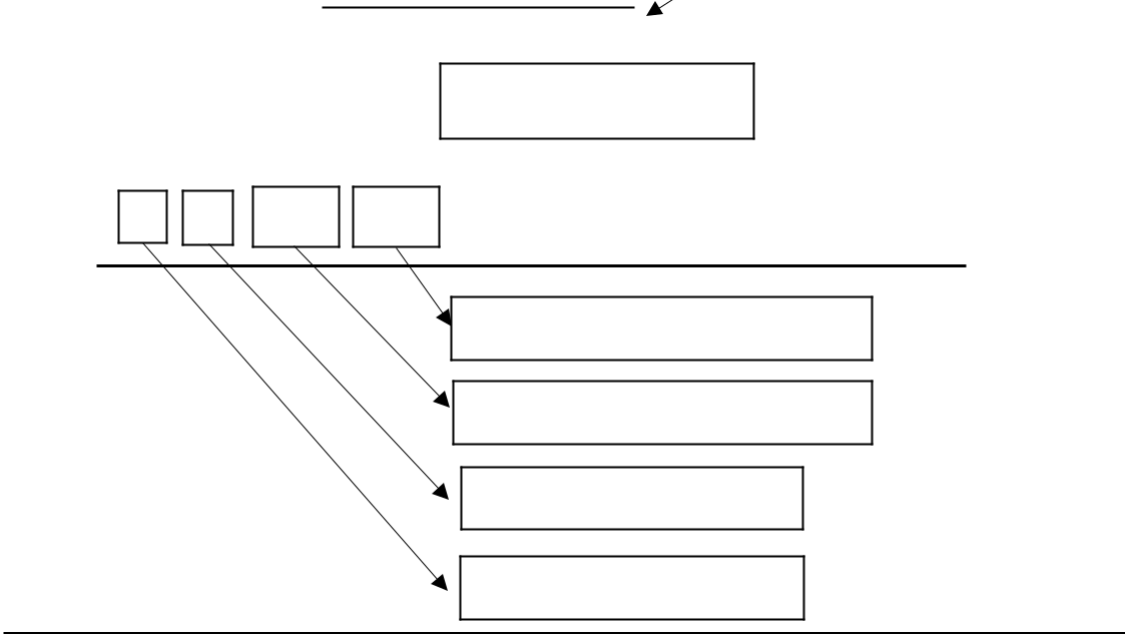
Многопользовательский режим представляет собой использование двумя игроками компьютерной мышки для выполнения ходов по очереди.

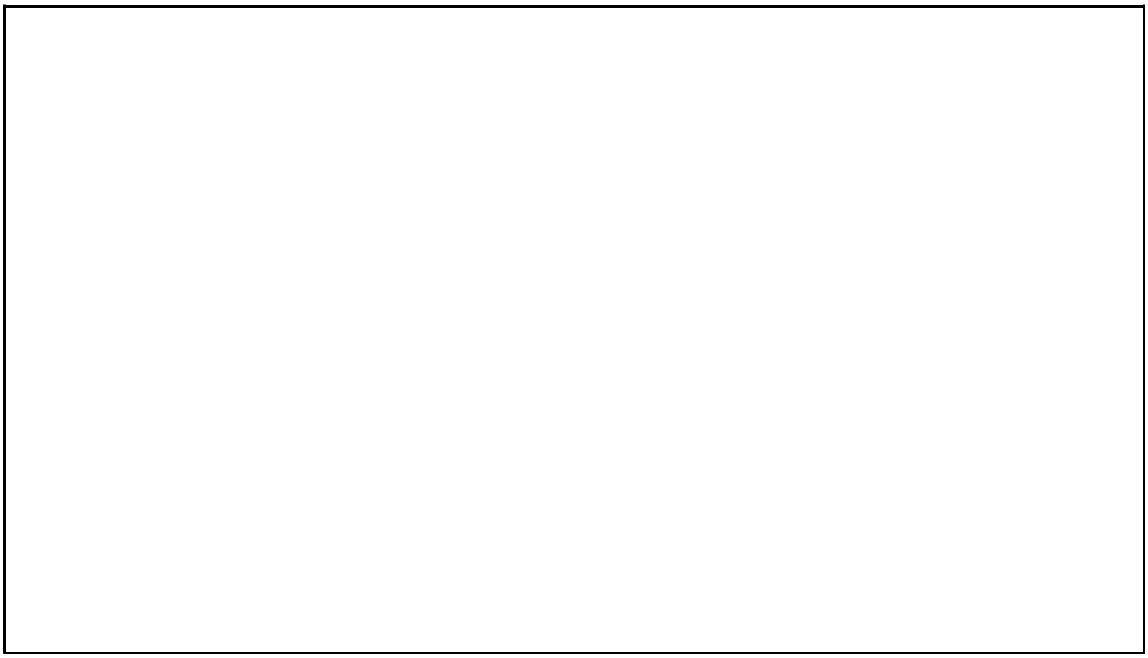
3.8. Интерфейс пользователя

3.8.1. Блок – схема

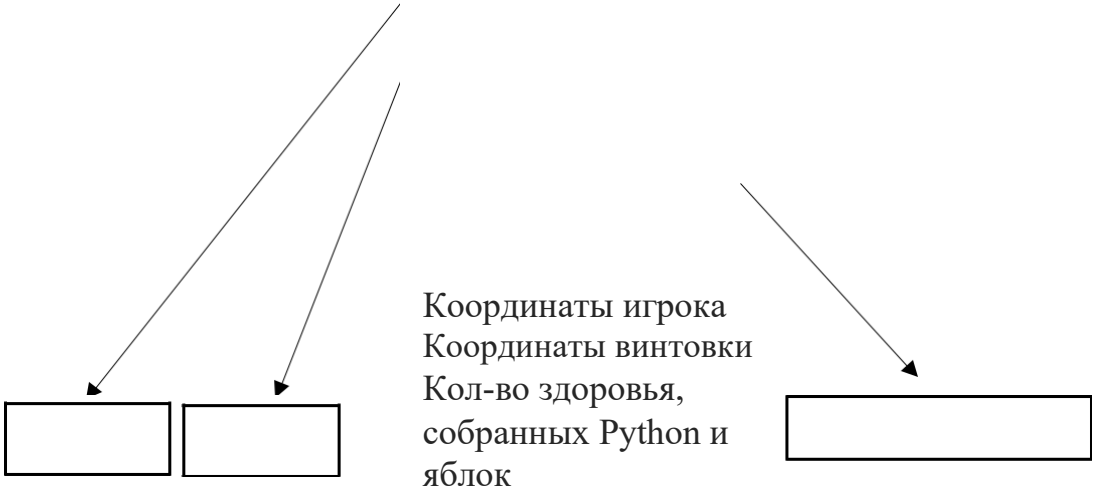
Интерфейс пользователя организован по следующим схемам:

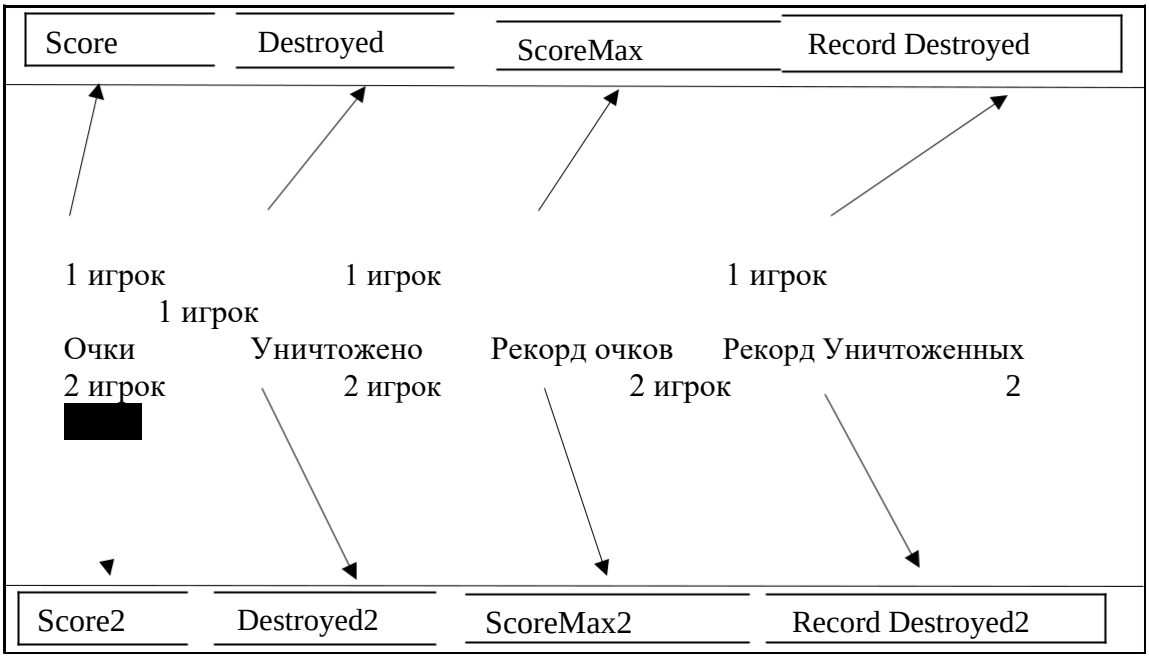




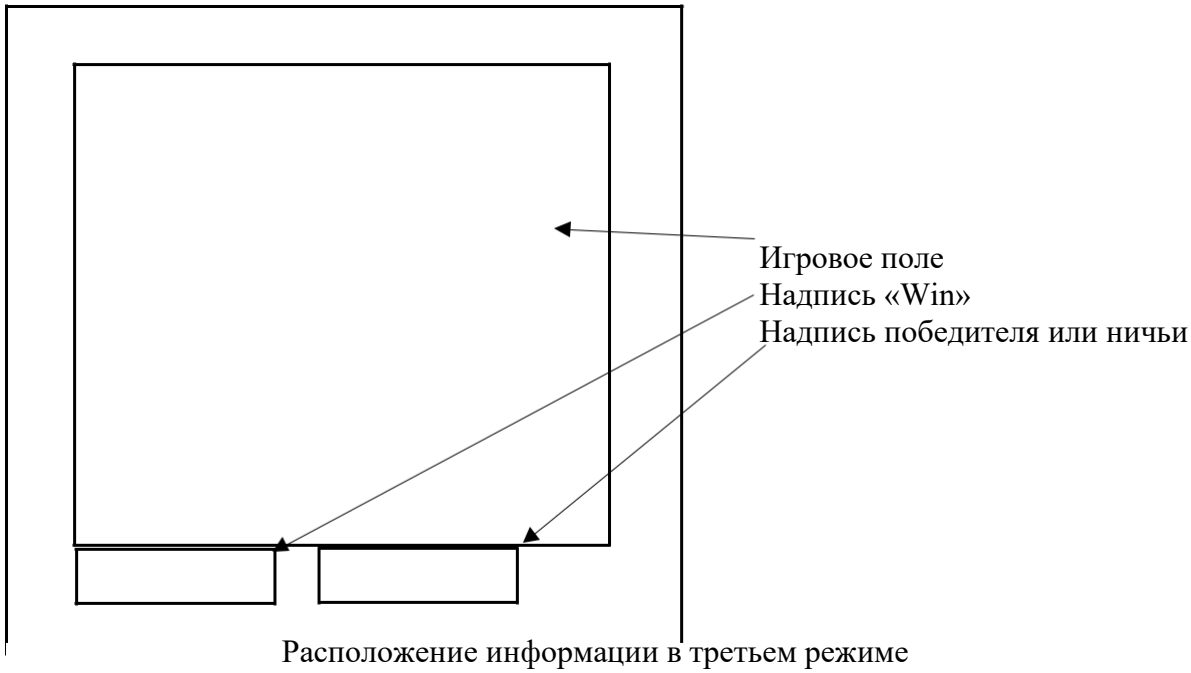


Расположение оконо информации в первом режиме





Расположение информации во втором режиме



Расположение информации в третьем режиме

Игровое пространство представляет собой классическую **2D** проекцию с горизонтальным положением камеры, которая следует за движением главного героя.

3.8.2.2. Второй режим

Игровое пространство представляет собой классическую **2D** проекцию с положением камеры сверху, которая не следует за движением главного героя.

3.8.2.3. Третий режим

Игровое пространство представляет собой классическую **2D** проекцию с горизонтальным положением камеры.

3.8.3. Функциональное описание и управления

Функционирование игрового меню не имеет каких-либо особенностей и не требует дополнительного описания.

3.8.3.1. Первый режим

Клавиша	Результат

Up	Передвижение вверх
Left	Передвижение влево
Right	Передвижение вправо
Down	Передвижение вниз
W	Восстановление здоровья на 100 единиц
Q	Положить оружие
R	Подобрать оружие
U	Отдаление камеры
I	Приближение камеры
J	Вернуть камеру в исходное положение
G	Создать Утку
T	Использовать яблоко
Mouse Left	Стрельба
Управление курсоров	Направление стрельбы винтовки

3.8.3.2. Второй режим

3.8.3.2.1. Первый игрок

Клавиша	Результат
Up	Движение вверх
Down	Движение вниз
Left	Движение влево
Right	Движение вправо
Space	Стрельба

3.8.3.2.2. Второй игрок

Клавиша	Результат
W	Движение вверх
S	Движение вниз
A	Движение влево
D	Движение вправо
C	Стрельба

3.8.3.3. Третий режим

Управление осуществляется только мышкой и левой кнопкой мышки – выбор клетки хода.

3.8.4. Объект интерфейса пользователя

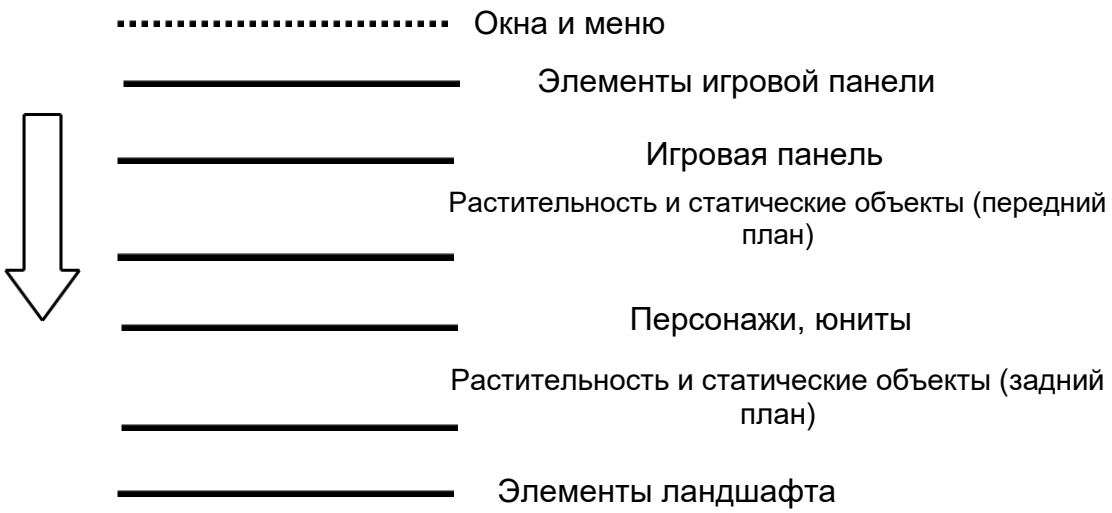
Объектами интерфейса пользователя являются:

Окно – графическая область прямоугольной формы с изображением и расположенными на ней другими объектами. Окнами являются menu, Quest log и другие.
Кнопка – графическая область прямоугольной или овальной формы с изображением и ассоциированной командой. Может использовать эффект вдавливания при нажатии, а может и нет.
Поле текста – прямоугольная область в рамках окна, содержащая форматированный.
Нефункциональная область – любая область на экране, взаимодействие с которой не дает никакого эффекта (например, информация о количестве ресурсов на игровом экране, края окон и др.).

3.9. Графика и видео

3.9.1. Общее описание

Изображение игры строиться на основе многослойного наложения различных элементов: элементов ландшафта, специальных объектов, игровых объектов, элементов интерфейса:



3.9.2. Двумерная графика и анимация

Для создания игры требуется разработка следующих основных графических частей:

Интерфейс
Персонажи, строения и юниты
Ландшафты Статические объекты

Интерфейс.

К графическим элементам интерфейса относятся:

Элемент	**Комментарии**
Шрифты игры	Основные шрифта игры, включающие латиницу
Окна Новая игра, FAQ, выход, увеличение громкости, уменьшение громкости, второй режим, третий режим	Изображения для расположения других функциональных элементов интерфейса
Прямоугольные кнопки	Изображения для реализации кнопок меню.
Курсоры	курсор – основной указательный

Игровая панель	Основная игровая панель

3.9.3. Ландшафты.

В игре предполагается создание ландшафтов, как приведено в следующих таблицах

3.9.3.1. Первый режим

Элемент	Комментарии
Трава	1 типов, без анимации
Булыжник	2 типа, без анимации
Плитка	1 тип, без анимации
Кирпич	1 тип, без анимации
Паутина	1 тип, без анимации

3.9.3.2. Второй режим

Элемент	Комментарии
Трава	1 тип, без анимации
Кирпич	1 тип, без анимации

3.9.3.3. Третий режим

Ландшафта не имеет.

3.10. Звуки и музыка

3.10.1. Общее описание

Звуковое и музыкальное оформление присутствует только в главном меню и в первом режиме. В меню музыка опирается на принцип динамической музыки, а в первом режиме на создание жуткой атмосферы.

3.10.2. Звуки

К звукам **объектов игры** относится:

- Стрельба из автоматической винтовки

3.10.3. Музыка

В игре предлагаются две музыкальных темы. Одна играет в меню, другая в первом режиме.

Музыкальные темы:

Тема из главного меню игры «Battlefield 3»

Тема из уровня «Мы не ходим в Рейвенхольм» из игры «Half-life 2»

3.11. Описание уровней

3.11.1. Общее описание уровней

3.11.1.1. Первый режим

Колледж – Квадратная карта с бледной травой и двумя зданиями. Одно маленькое с одним входом. Второе здание в несколько раз больше и имеет два входа, широкий коридор и одну большую комнату. Повсюду на карте встречается паутина, а также можно найти два ж/д пути.

3.11.1.2. Второй режим

Окраина города – квадратная карта с бледной травной и руинами кирпичных зданий. Представляет собой бессистемный лабиринт.

3.11.1.3. Третий режим

Особых уровней не имеет.

4. Контакты

Контактные лица: Стрельников Сергей

E-mail: genstrser@gmail.com