Государственное бюджетное образовательное учреждение высшего образования
Московской области
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка компьютерной игры-платформера типа “Марио” на C++»

Выполнил студент

Растопчин Андрей Романович

Группа П2-17

_____(Подпись)

_____ (Дата сдачи работы)

Проверил преподаватель

Гусятинер Леонид Борисович

_____(Подпись)

_____(Оценка)

Королёв 2020 г.

Оглавление

Введение

Данный курсовой проект посвящен созданию 2D игры на языке C++.

Выбору такого жанра компьютерной игры, как платформер, послужили: возможность понимания основных механик создания игры и получение полезного опыта при работе с визуальной составляющей.

Цель курсового проекта заключается в понимании и использовании основных методов создания 2D-игр, изучения свободной, кроссплатформерной и мультимедийной библиотеки SFML, а также полезный опыт работы со средой разработки Microsoft Visual Studio.

Для осуществления поставленной задачи служат следующие пункты:

1. Изучение литературы, документации и программных решений по SFML и C++;
2. Подбор мультимедии и подходящих шаблонов для создания будущего персонажа;
3. Написание кода;
4. Описание руководства оператора.

Глава 1. Теоретическая часть

1.1 Описание предметной области

Платформер - жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня.

Противники (называемые «врагами»), всегда многочисленные и разнородные, обладают примитивным искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает очки жизни у героя или вовсе отправляет в начало уровня. Чтобы справиться с противником обычно достаточно обойти его (если это необязательный персонаж) или устранить его прыгнув ему на голову или из оружия, если оно есть. Устранение противника обычно изображается символически (существо исчезает или проваливается под карту).

Уровни как правило наполнены большим количеством потайных (невидимых) комнат или бонусами, что упрощает прохождение и подогревает к нему интерес.

Игры такого жанра как правило характеризуются мультяшностью окружения и персонажей.

Путь платформеров начался в 1980-х годах с 2D графики, когда игровые консоли не были достаточно мощными, чтобы отображать трехмерную графику или видео. Они были ограничены статическими игровыми мирами, которые помещались на один экран, а игровой герой был виден в профиль. Персонаж лазал вверх и вниз по лестницам или прыгал с платформы на платформу, часто сражаясь с противниками и собирая предметы, улучшающие характеристики. Первыми играми этого типа были *Space Panic* и его

последователь *Donkey Kong*, эти игры относились к аркадному жанру (Рис. 1.1. Игра Donkey Kong).

Рис. 1.1. Игра Donkey Kong.

Вскоре процесс прохождения уровня перестал быть в основном вертикальным и стал горизонтальным с появлением длинных многоэкранных прокручивающихся игровых миров. Считается, что начало этому положила выпущенная фирмой Activision в 1982 году игра *Pitfall!* (Рис. 1.2. Pitfall!) для консолей Atari 2600. Стоит отметить, что особенностью игры являясь резкая смена кадров при перемещении по карте, а не всем так примычное следование камеры за игровым персонажем.

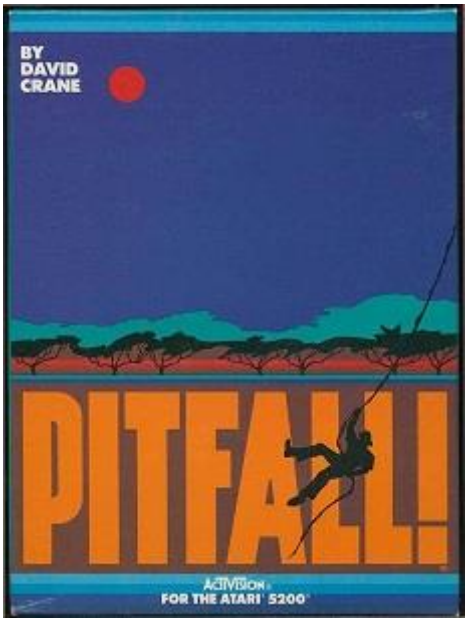

Рис. 1.2. Игра Pitfall!

В 1985 году фирма «Nintendo» выпустила для приставки Nintendo Entertainment System революционный платформер Super Mario Bros. (Рис. 1.3 Игра Super Mario Bros.)

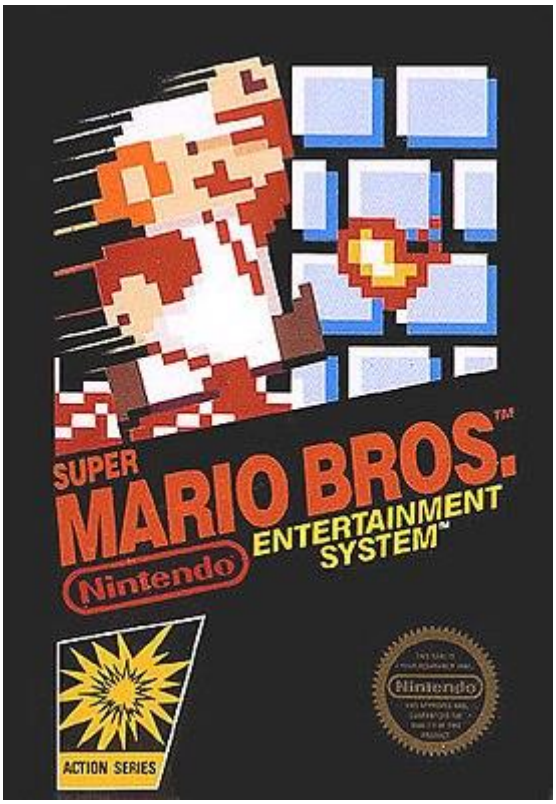

Рис. 1.3. Игра Super Mario Bros.

Игра была наполнена большими и сложными уровнями, и стала примером для последующих создателей игр, и даже сегодня многие люди считают её одной из самых лучших видеоигр.

Трёхмерный платформер - геймплей, включающий все три измерения, или использование трёхмерных полигонов в реальном времени для отрисовки уровней и героев, или и то и другое. Появление трёхмерных платформеров принесло изменение конечных целей некоторых платформеров. В большинстве двумерных платформеров игроку нужно было достичь на уровне только одной цели, однако во многих трёхмерных платформерах, каждый

уровень необходимо прочесывать, собирая кусочки головоломок (*Banjo-Kazooie*) или звезды *Super Mario 64* (Рис. 1.4. Игра Super Mario 64.).

Рис. 1.4. Игра Super Mario 64.

Изометрические платформеры - поджанр и дву-, и трёхмерных платформеров. Они отображают трёхмерную сцену с помощью двумерной графики, которая отображает мир с жёстко ориентированной камеры без учёта перспективы. И хотя изометрические платформеры не были первыми изометрическими играми, ранними примерами изометрических платформеров в мире аркадных автоматов является игра 1983 года *CongoBongo* (Рис. 1.5. Игра Congo Bongo.)

Рис. 1.5. Игра Congo Bongo.

Super Mario - серия компьютерных игр в жанре платформер, издаваемых компанией Nintendo. Часть медиафраншизы Mario. Первая игра в серии - *Super Mario Bros.* - вышла в 1985 году, последняя - *Super Mario Maker 2* - в 2019 году. Серия *Super Mario* является основной линейкой игр в своей франшизе.

Особо можно отметить *Super Mario 64*, эта часть стала первой игрой с трехмерной графикой во франшизе.

1.2 Описание существующих разработок

Jet Fire - игра жанра платформер основной целью которой является увеличение пройденного расстояния (Рис. 1.6. Игра Jet Fire).

Игра вызывает тягу к получению все новых и новых рекордов, а возможность улучшения персонажа посредством сбора монет во время игры еще больше подкрепляет интерес к игре.

Из улучшений программной составляющей хотелось бы предложить увеличение скорости персонажа, более резкие падение и прыжок, а также возможность перемещения персонажа вперед, но и назад.

Рис. 1.6. Игра Jet Fire.

Ссылка на источник: http://flashdozor.ru/play-53367.html

Toto Double Trouble - игра жанра платформер и представляет собой игру сделанную исключительно для 2 игроков, целью которых является вместе дойти до конца уровня (Рис. 1.7. Игра Toto Double Trouble).

Персонажи полностью зависят друг от друга и ни один, ни второй не смогут дойти до конца без взаимопомощи.

Плюсом игры является необходимость обдумывания своих действий и работа в команде, а также сложность, которая растет с каждым новым уровнем.

А из минусов стоит отметить необходимость нахождения второго игрока непосредственно за одним компьютером с первым, так как персонажи управляются с одного компьютера.

Рис. 1.7. Игра Toto Double Trouble.

Источник игры: http://flashdozor.ru/play-53360.html

Глава 2. Проектная часть

2.1 Диаграмма прецедентов

Происходит взаимодействие игрока с персонажем (Mario).

В свою очередь персонаж взаимодействует с объектами и другими персонажами (Рис. 2.1. Диаграмма прецедентов):

1. Взаимодействие с блоком выдающем монеты приводит к получению очков, которые прибавляются к счету игрока.
2. Взаимодействие с “Противником” приводит к его устранению и как следствию получения очков.
3. Взаимодействие с блоком выдающем грибы приводит к подбору гриба персонажем игрока и временному усилению персонажа (неуязвимости).

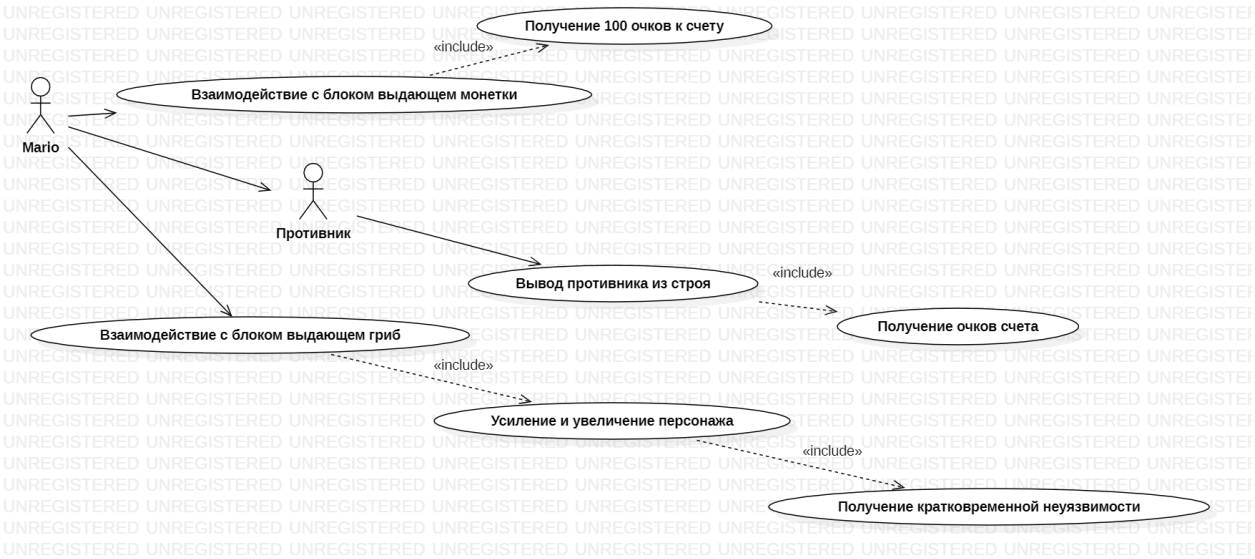

Рис. 2.1. Диаграмма прецедентов.

2.2 Выбор инструментов

При выборе инструментов было проведено сравнение по критериям представленных в таблице 1.
За каждый критерий давалась оценка от 0 до 10 баллов за критерий.

Таблица 1: Критерии выбора и их оценка.

Критерии	**CodeBlocks**	**Microsoft Visual Studio**
Участие в корпоративном проекте	8	10
Простота сопровождения	10	8
Наличие библиотек	7	10
Наличие документации на русском языке	6	10
Скорость разработки и т.д.	8	9
Итого:	39	47

Исходя из составленных оценок, был выбран Microsoft Visual Studio.

2.3 Проектирование сценария

Работа сценария строится следующим образом (Рис. 2.2. Сценарий):

1. Происходит соединение с программой (запуск);
2. Дальше мы попадаем в главный модуль, от куда уже осуществляется работа с персонажами игры (Игрок и Противник) и картой;
3. При работе с персонажем “Игрок” происходит его отрисовка, проверка на столкновения, и реализация движения персонажа по кадрам.
4. При работе с персонажем “Противник” происходит все тоже самое, что и с персонажем “Игрок”;
5. Работая с модулем “Карта” происходит создание макета, по которому в дальнейшем, будет визуализирована карта.

Рис. 2.2. Сценарий.

2.4 Диаграмма классов

В диаграмме представлены два класса PLAYER (Игрок) и ENEMY(противник).

В классе PLAYER используется спецификатор доступа public, что говорит нам о том, что доступ открыт всем, кто видит определение данного класса.

Блок стоящий после PLAYER показывает нам объявленные переменные, а знак “+” указывает на спецификацию доступа, в данному случае только public.

После блока с атрибутами, расположен блок с операциями (функциями класса).

У класса ENEMY структура полостью совпадает со структурой класса PLAYER, за исключением еще одной операции (Установка()).

Рис. 2.3. Диаграмма классов.

2.5 Описание главного модуля

Главный модуль включает в себя несколько частей:

1. Создание текста;
2. Создание звука для прыжка;
3. Меню (Листинг 3);
4. Фоновая музыка;
5. Главный цикл программы с временем и коллизией персонажей;
6. Перевод макета карты в визуальную составляющую (Листинг 4).

Создание текста. Получаем файл шрифта и определяем ему формат.

Листинг 1. Создание текста.

```
Font font; // Шрифт
// Передаем нашему шрифту файл шрифта
font.loadFromFile("CyrilicOld.ttf");
// Создаем объект текст. закидываем в объект
// текст строку, шрифт, размер шрифта(в пикселях),
// сам объект текст (не строка)
```

```
Text text("", font, 15);
// Покрасили текст в красный
text.setFillColor(Color::Red);
text.setStyle(Text::Bold); // Жирный текст.
```

Создание звука для прыжка. Получаем файл звука в буфер и определяем, как звук. При каждом прыжке вызываем этот звук.

Меню. Заключаем картинки с пунктами меню в текстуры и создаем на их основе спрайты, определяем пункт меню (Листинг 2) по наведению мышки и при нажатии совершаем действие в соответствии с пунктом меню (Листинг 3).

Листинг 2. Выбор пункта меню.

```
//При наведении курсора мышки на поле в рамках указанных координат
if (IntRect(100, 30, 300, 50).contains(Mouse::getPosition(window)))
{
  menu1.setColor(Color::Yellow); //Пункт меню окрашивается в желтый
  MenuNum = 1; //Указываем номер пункта меню
}
```

Листинг 3. Действия меню при выборе пункта

```
if (Mouse::isButtonPressed(Mouse::Left))
{
  // Если указан первый пункт "Новая игра",
  // меню прекращает работу и запускается игра
  if (MenuNum == 1)
    Menu = false;
  // При выборе второго пункта меню "О программе".
  if (MenuNum == 2)
  {
    // Заготавливается картинка с описанием
    window.draw(about);
    window.display(); // Выводится на экран
    // Картинка с описанием не закроется пока
    // не будет нажата кнопка "Escape".
    while (!Keyboard::isKeyPressed(Keyboard::Escape));
  }
```

Фоновая музыка. Файл с музыкой записывается в переменную определенную классом Music и в отличии от звука играет на протяжении всей игры (времени самой музыки).

Главный цикл программы с временем и коллизией персонажей. В этой части определенной бесконечным циклом while (window.isOpen()), происходит создание времени, на которое завязано обновление всех действий в игре, определена коллизия (проверка на столкновение) “Героя” и “Врага”, а также создана привязка действий персонажа “Герой” к кнопкам.

Перевод макета карты в визуальную составляющую. Эта часть программы определена циклом for который проходит по всем элементам массива (макета карты) и определяет на места ключей выбранные с помощью метода Прямоугольника IntRect(99, 224, 140 - 99, 255 - 224) картинки.

Листинг 4. Визуализация карты по макету.

```
for (int i=0; i < H; i++)
for (int j=0; j < W; j++)
{
  // Если в массиве встречен символ P, то по заданным
  // координатам с картинки, формируется моделька текстуры
  if (TileMap[i][j] == 'P')
    tile.setTextureRect(IntRect(143-16 * 3, 112, 16, 16));
```

2.6 Описание спецификаций к модулям

Спецификация к модулю PLAYER (Листинг 5).

Листинг 5. Спецификации модуля PLAYER.

```
#ifndef PLAYER_H
#define PLAYER_H
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <sstream>

using namespace sf;

class Player
{
public:

  float dx, dy; // Переменные сдвига по оси x и y
```

```
  // Переменная для заключения нашей картинки в прямоугольник.
  FloatRect rect;
  // Переменная для определения нахождения персонажа на земле.
  bool onGround;
  // Переменная для объединения текстуры с прямоугольником
  Sprite sprite;
  // Переменная для хранения текущего кадра
  //и отрисовки анимации персонажа
  float currentFrame;

  Player(Texture &image);
  void Update(float time, float offsetX, float offsetY);
  void Collision(int num);
};
#endif // PLAYER_H
```

Спецификации к модулю ENEMY (Листинг 6).

Листинг 6. Спецификации к модулю ENEMY.

```
#ifndef ENEMY_H
#define ENEMY_H
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <sstream>

using namespace sf;

class Enemy
{

public:
  float dx, dy; // Переменные смещения по осям X и Y.
  FloatRect rect; // Переменная для заключения картинки в прямоугольник
  Sprite sprite; // Переменная для объединения текстуры с прямоугольником
  // Переменная хранения в ней текущего кадра и отрисовки анимации персонажа
  float currentFrame;
  bool life; // Переменная определяющая состояние противника.

  void Set(Texture &image, int x, int y);
  void Update(float time, float offsetX, float offsetY);
  void Collision();
};
#endif // ENEMY_H
```

2.7 Описание модулей

Функции модуля Player:

1. Player
2. Update
3. Collision

Конструктор Player (Листинг 7). Этот конструктор модельку персонажа. В конструкторе определяется спрайт персонажа, рамка персонажа, ускорение по осям X и Y, а также задается начало кадра анимации персонажа.

Листинг 7. Конструктор Player.

```
Player::Player(Texture &image)
{
	// Определяем передаваемую ранее текстуру в спрайт.
	sprite.setTexture(image);
	// Задаем прямоугольник, x, y координаты верхней левой
	// координаты прямоугольника, остальные это ширина и высота
	rect = FloatRect(100, 180, size,size);
	dx = dy = 0.1;
	currentFrame = 0;
}
```

Функция Update (Листинг 8). Функция получает на вход время и смещения по осям X и Y. Функция осуществляет вызов функции Collision определяет состояние персонажа на земле и в прыжке и реализует смену кадров при движении.

Лтстинг 8. Функция обновления.

```
// Эта функция осуществляет синхронизацию со временем,
// реализацию функции коллизии, а также анимации персонажа.
void Player::Update(float time, float offsetX, float offsetY)
{
	rect.left += dx * time;
	Collision(0);
	if (!onGround)
		dy += 0.0005 * time; // Скорость (высота) прыжка вверх
	rect.top += dy * time;
	onGround = false;
	Collision(1);
```

```
      //Привязка кадра по времени к программе
      currentFrame += time * period;
      //Условие смены кадров (картинок всего 3, меняется положение ног)
      if (currentFrame > 3)
            currentFrame -= 3;
      if (dx > 0)
      //Осуществление смены кадров при движении персонажа вперед
      sprite.setTextureRect(IntRect(112 + 31 * int(currentFrame), 144, size,
      size));
      if (dx < 0)
      //Осуществление смены кадров при движении персонажа
      //назад (определено инверсией от движения вперед)
      sprite.setTextureRect(IntRect(112 + 31 * int(currentFrame) +size, 144,
      size * -1, size));
      //Определение позиции появления персонажа
      sprite.setPosition(rect.left - offsetX, rect.top - offsetY);
      dx = 0;
}
```

Функция Collision (Листинг 9). На вход функция получает значение нахождения персонажа в воздухе или на земле Collision(num). Функция отвечает за столкновение персонажа с объектами карты из разных положений.

Листинг 9. Коллизия.

```
void Player::Collision(int num)
{
  for (int i = rect.top / size; i < (rect.top + rect.height) / size; i++)
    for (int j = rect.left / size; j < (rect.left + rect.width) / size; j++)
    {
         //Условие нахождения различных объектов на карте.
         if ((TileMap[i][j] == 'P') || (TileMap[i][j] == 'k') || (TileMap[i][j]
         == '0') || (TileMap[i][j] == 'r') || (TileMap[i][j] == 't') ||
         (TileMap[i][j] == 'c'))
         {
           // Определение верхней координаты прямоугольника, при
           // условии нахождения персонажа в воздухе
           // и движения вниз.
           if (dy > 0 && num == 1)
           {
                 rect.top = i * size - rect.height;
                 dy = 0;
                 onGround = true;
           }
           // Определение верхней координаты
           // прямоугольника, при
           // условии нахождения персонажа
           // в воздухе и движения вверх.
           if (dy < 0 && num == 1)
           {
                 rect.top = i * size + size;
```

```
                dy = 0;
            }
            // Определение левой координаты
            // прямоугольника, при
            // условии движения персонажа вперед и по земле.
            if (dx > 0 && num == 0)
            {
                rect.left = j * size - rect.width;
            }
            // Определение левой координаты
            // прямоугольника, при
            // условии движения персонажа назад и по земле.
            if (dx < 0 && num == 0)
            {
                rect.left = j * size + size;
            }
        }
    }
}
```

Функции модуля Enemy:

1. Set
2. Update
3. Collision

Функция Set (Листинг 10). Функция принимает на вход модельку персонажа “Противника” и координаты X, Y Set(Texture &image, int x, int y).

Функция создает рамку, спрайт персонажа, определяет скорость передвижения, задает начало смены кадров и определяет состояние персонажа “Противник” (жив или побежден).

Листинг 10. Установка “Противника”.

```
void Enemy::Set(Texture &image, int x, int y)
{
    sprite.setTexture(image);
    rect = FloatRect(x, y, size, size);
    dx = 0.05;
    currentFrame = 0;
    life = true;
}
```

Функция Update (Листинг 11). Функция получает на вход время и смещения по осям X и Y. Функция осуществляет вызов функции Collision определяет состояние персонажа на земле и в прыжке и реализует смену кадров при движении.

Листинг 11. Обновление “Противника”.

```
// Эта функция осуществляет синхронизацию со временем,
// реализацию функции коллизии, а также реализует анимацию персонажа.
void Enemy::Update(float time, float offsetX, float offsetY)
{
      rect.left += dx * time;
      Collision();
      currentFrame += time * period;
      if (currentFrame > 2)
            currentFrame -= 2;
      // Определяет анимацию противника.
      sprite.setTextureRect(IntRect(18 * int(currentFrame), 0, size, size));
      // Определяет анимацию противника после победы героя над ним.
      if (!life) sprite.setTextureRect(IntRect(58, 0, size, size));
      // Установка спрайта противника по заданным координатам.
      sprite.setPosition(rect.left - offsetX, rect.top - offsetY);
}
```

Функция Collision (Листинг 12). Функция проверяет столкновение модельки “Противника” с трубами и осуществляет блуждание.

Листинг 12. Коллизия “Противника”.

```
// Функция проверки столкновения противника с трубами
// и нахождения на земле.
void Enemy::Collision()
{
  // Цил проходящий по объектам карты
  for (int i = rect.top / size; i < (rect.top + rect.height) / size; i++)
    for (int j = rect.left / size; j < (rect.left + rect.width) / size; j++)
        // Условие нахождения объекта
        // на карте с соответствующими ключами
        if ((TileMap[i][j] == 'P') || (TileMap[i][j] == '0'))
        {
          if (dx > 0) // Условие движения противника вправо.
          {
                // Определение точки столкновения
                rect.left = j * size - rect.width;
                // Осуществляет блуждание персонажа
                // вперед и назад,
```

```
                    // отталкивающегося от объектов.
                    dx *= -1;
                }
                // Условие движения персонажа влево.
                else if (dx < 0)
                {
                rect.left = j * size + size;
                dx *= -1;
                }
            }
}
```

2.8 Описание тестовых наборов модулей

Здесь будет представлен результат тестирования “Белого ящика”

Тест 1. Изменение текста счетчика прыжков.

Чтобы изменить текст, необходимо открыть проект и в файле main.cpp перейти к строке 267 (Рис. 2.4).

```
264         playerScoreString << countJump;
265         // Задаем строку тексту и вызываем
266         // сформированную выше строку
267         text.setString("Сделано прыжков:" + playerScoreString.str());
268         // Задаем позицию текста, отступая от центра камеры
```

Рис. 2.4 Надпись.

И меняем ее на более простое: “Прыжки” (Рис. 2.5).

```
265         // Задаем строку тексту и вызываем
266         // сформированную выше строку
267         text.setString("Прыжки:" + playerScoreString.str());
268         // Задаем позицию текста, отступая от центра камеры
```

Рис. 2.5 Надпись v.2.

Результат:

Было:

Рис. 2.6. Было.

Стало:

Рис. 2.7. Стало.

Тест 2. Изменение и добавление объектов на карту.

Для данного теста, необходимо открыть файл map.h и перейти к 24 строке (Рис. 2.7).

```
23    "0
24    "0                    c     kckck
25    "0
```

Рис. 2.8. Макет карты.

Изменим строку следующим образом (Рис. 2.8):

```
23    "0
24    "0                      ccccckkkkk
25    "0
```

Рис. 2.9 Макет карты v.2.

Результаты теста:

Было:

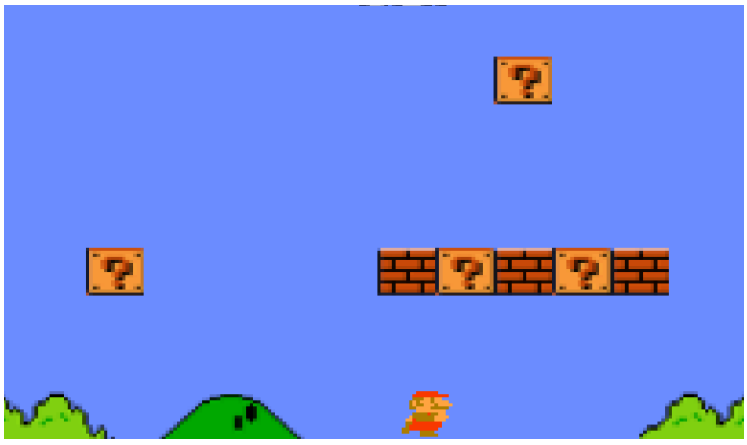

Рис. 2.10 Ранняя версия карты.

Стало:

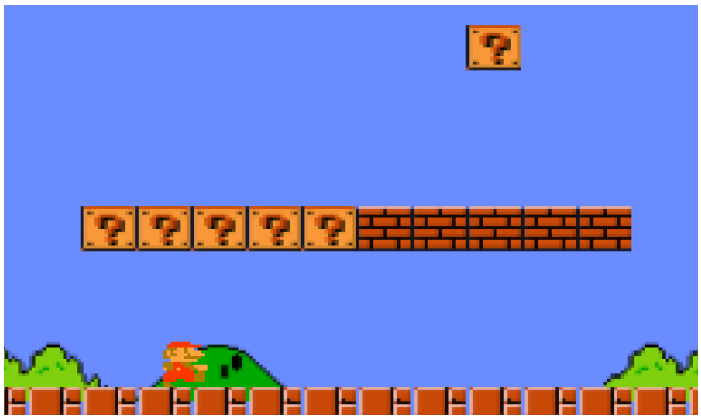

Рис. 2.11 Новая версия карты.

2.9 Описание применения средств отладки

В ходе написания курсового проекта было получено следующее сообщение:

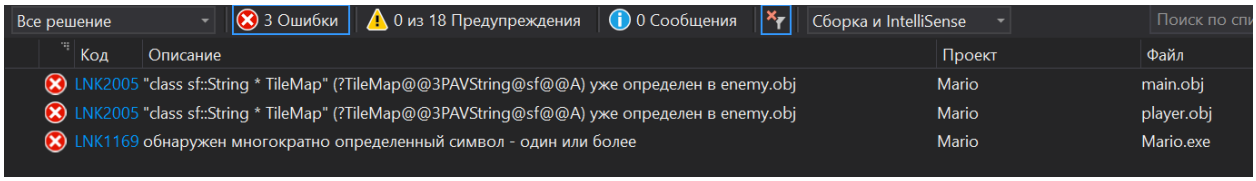

Рис 2.12 Ошибка обнаружение.

Суть ошибки заключается в многократном определении массива TileMap.

Для решения данной ошибки, необходимо перейти в файл map.h и обратить внимание на 13 строку (Рис. 2.11).

```
12    // будет собрана визуальная реализация карты из картинок.
13    String TileMap[H] = {
14    "0000000000000000000000000000000000000000000000000000000000000000000000
15    "0
```

Рис. 2.13 map.h.

Необходимо изменить строку `String TileMap[H] = {` на строку `const String TileMap[H] = {`, тем самым указав, что массив неизменяем.

Результатом отладки стало следующее сообщение:

```
Поток 0x3d04 завершился с кодом 0 (0x0).
Поток 0xbd4 завершился с кодом 0 (0x0).
Программа "[14820] Mario.exe" завершилась с кодом 0 (0x0).
```

Рис. 2.14 Результат отладки программы.

2.10 Анализ оптимальности использования памяти и быстродействия

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

Список принятых оптимальных решений:

1. В ходе оптимизации кода, было принято решение часто используемые переменные отдельных модулей сделать локальными, а не передавать из главного модуля. Пример такой оптимизации, представлен в Листинге 13.

2. При работе с блоком меню, было принято решение использовать картинки с текстом, а не текст, написанный самой программой, тем самым были убраны модули определения текстовых переменных и импортирование шрифтов. Пример такой оптимизации показан в Листинге 14.

Листинг 13.

```
#include "enemy.h"
#include "map.h"

// Ширина, и высота используемой рамки
const int size = 16;
// Степень уменьшения скорости анимации
const double period = 0.005;
```

Листинг 14.

```
Texture menu_texture1, menu_texture2;
```

```
Texture menu_texture3, about_texture;
menu_texture1.loadFromFile("111.png"); // Загрузка картинок в текстуры
menu_texture2.loadFromFile("222.png");
menu_texture3.loadFromFile("333.png");
about_texture.loadFromFile("about.png");
```

Глава 3. Эксплуатационная часть

3.1 Руководство оператора

3.1.1 Назначение программы

Функциональное назначение программы

Основной функцией проекта “Mario” является развлекательная составляющая. Игра помогает отвлечься от реального мира и погрузиться в увлекательное прохождение уровней и сражения с противниками.

Эксплуатационное назначение программы

Основное назначение проекта “Mario” – показать концепцию всех игр жанра платформер, визуализировать методы работы с текстурами, звуками и текстом, а также реализовать связь между созданными моделями и действиями игрока.

Состав функций

Эмоциогенная функция

Игра меняет эмоциональное состояние человека, как правило в лучшую сторону, повышает настроение и провоцирует рост воодушевления. Игра пробуждает целый спектр чувств у пользователя, это и удовольствие, и радость и даже страх.
Источник: http://tambolia.ru

Терапевтическая функция

Согласно с выводами Берна, в игры чаще всего играют люди с нарушенным душевным равновесием, игры помогают в снятии стресса и отвлечения от реальных проблем, чтобы было время расслабиться и вернуться к ним с новыми силами.

Источник: http://tambolia.ru

3.1.2 Условия выполнения программы

Минимальный состав аппаратных средств

Минимальный состав используемых технических (аппаратных) средств:

- Windows 7, Windows 8.1 или Windows 10;
- Процессор с тактовой частотой не ниже 1,8 ГГц.
- 2 ГБ ОЗУ; рекомендуется 8 ГБ ОЗУ
- Место на жестком диске: до 200 ГБ (20 – 50 ГБ для Microsoft Visual Studio 2017, 100 МБ проект);
- Видеоадаптер с минимальным разрешением 720p.

Минимальный состав программных средств

Системные программные средства, используемые проектом “Mario”, должны быть представлены локализованной версией операционной системы Windows 7, Windows 8.1 или Windows 10.

Также для функционирования проекта “Mario” на ПК необходимо предустановленное программное обеспечение стороннего разработчика, программа Microsoft Visual Studio 2017 и пакет библиотеки SFML, для 32-х и 64-х разрядной системы.

Требование к персоналу (Пользователю)

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

3.1.3 Выполнение программы

Загрузка и запуск программы

Загрузка и запуск проекта “Mario” осуществляется в следующем порядке.

Необходимо скачать проект с диска и распаковать его в папку.

Запускаем файл Mario.sln через Microsoft Visual Studio 2017. (Рис. 3.1).

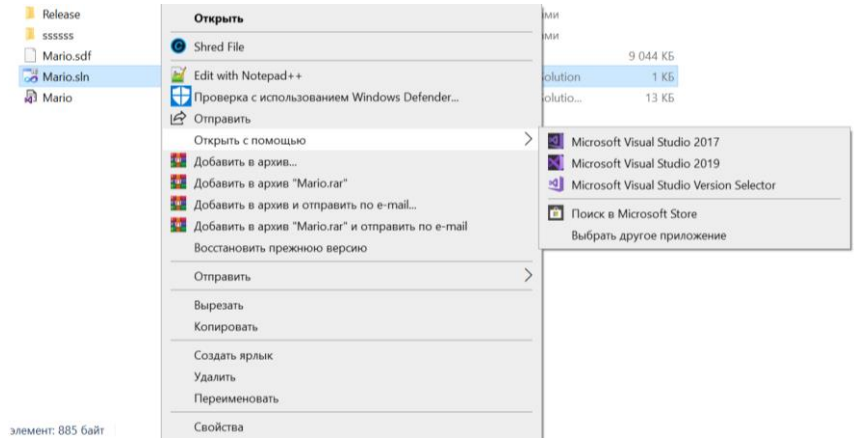

Рис. 3.1 Запуск проекта.

Для корректной работы проекта необходимо скачать библиотеку SFML (32-х битная версия). Нам необходима следующая версия файла (Рис. 3.2).

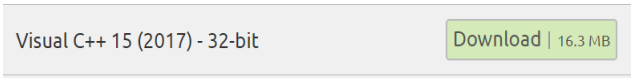

Рис. 3.2 Необходимые версии.

После скачивания мы получаем следующий результат (Рис. 3.3):

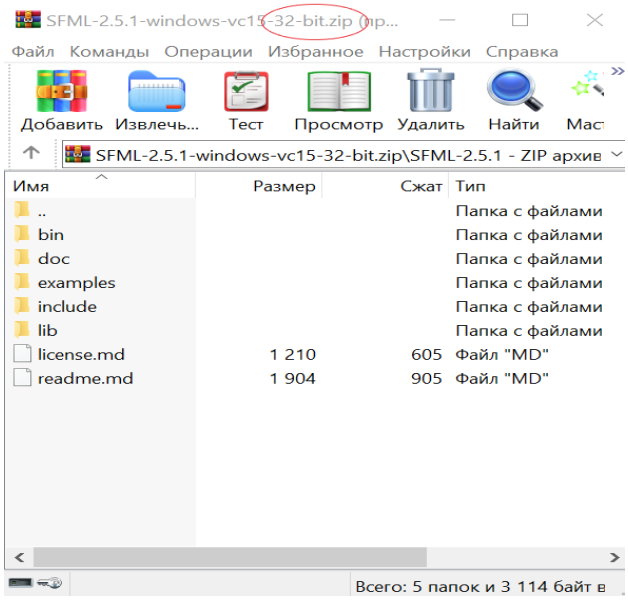

Рис. 3.3. Установленные версии.

Из этого файла нам понадобится только bin, include и lib (Рис. 3.4).

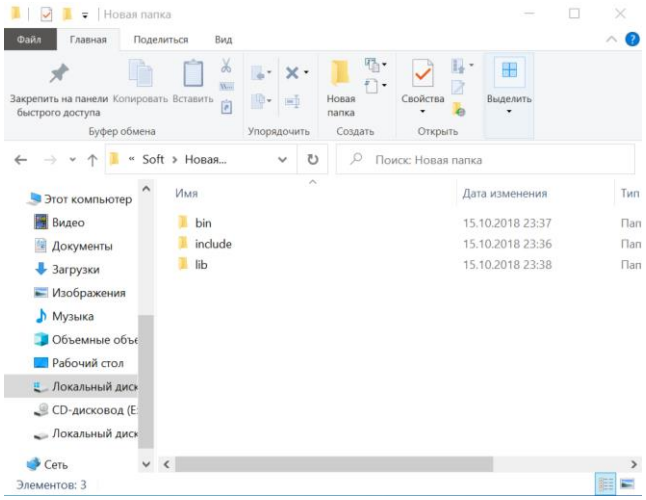

Рис. 3.4 Общая папка.

Далее необходимо вернуться к открытому проекту и перейти в свойства проекта (Рис 3.5).

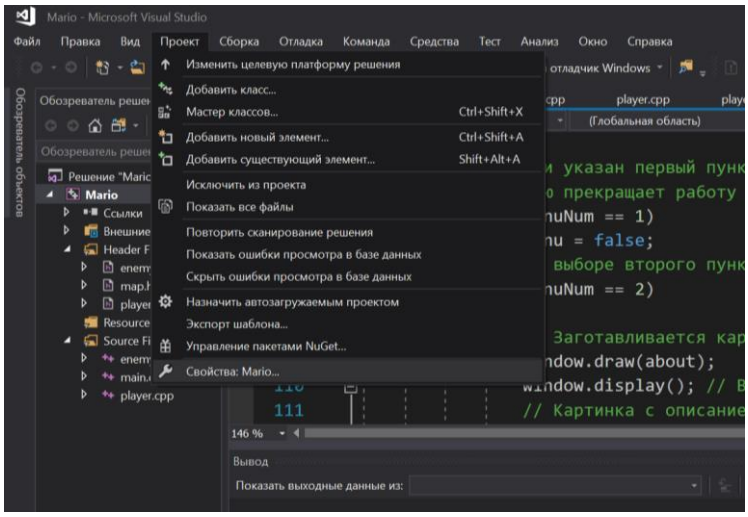

Рис. 3.5. Свойства проекта.

Необходимо выбрать конфигурацию “Все конфигурации” и поменять путь к дополнительным каталогам включаемых файлов на путь к вашему include, который ранее вы положили в общую папку (Рис. 3.6).

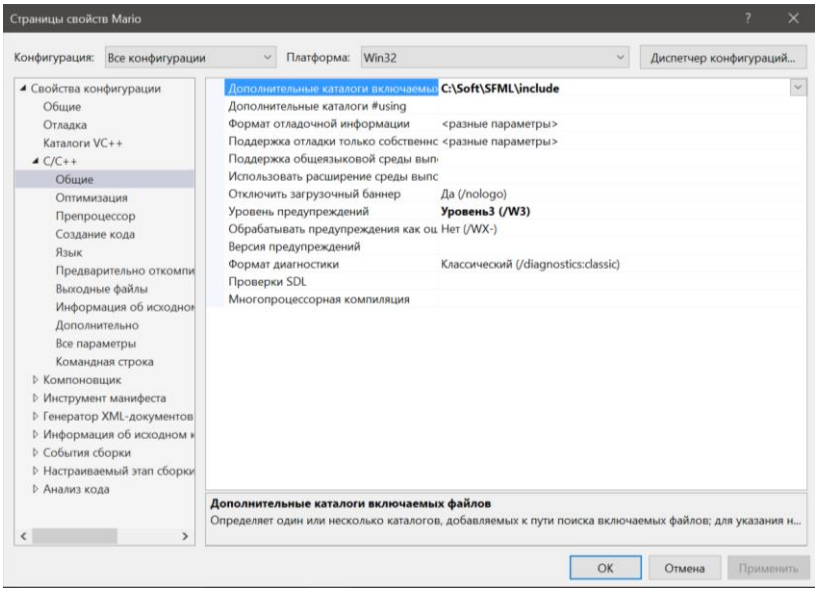

Рис. 3.6 Include.

Далее необходимо сменить конфигурацию на “Debug” и изменить путь к дополнительным каталогам библиотек, на путь к вашему lib (Рис. 3.7).

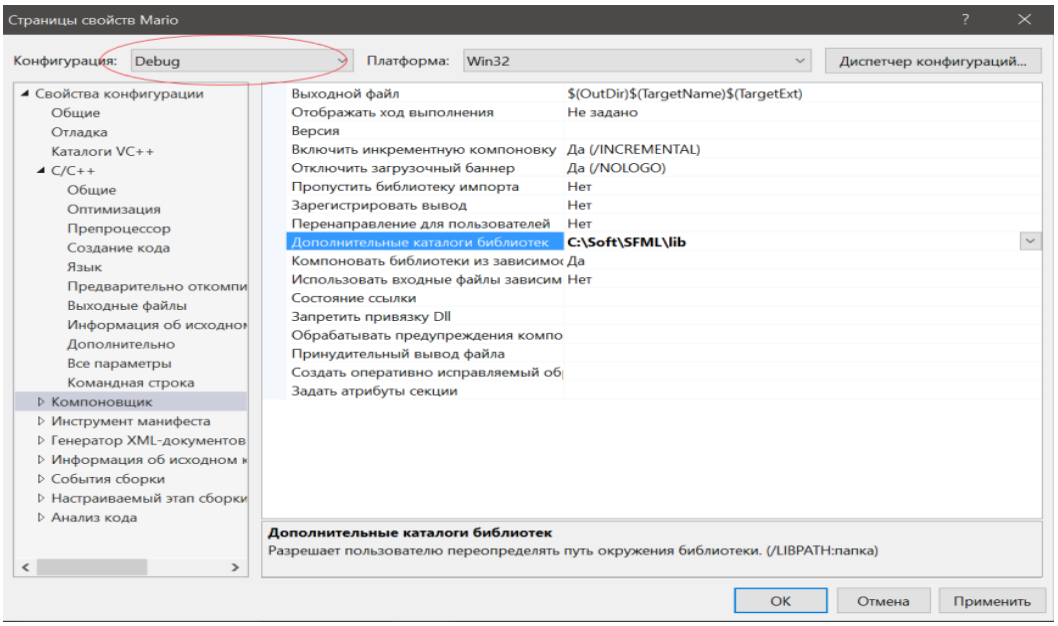

Рис. 3.7 Debug.

Следующим действием меняем конфигурацию на Realese и повторяем действия, как с Debug (Рис. 3.8).

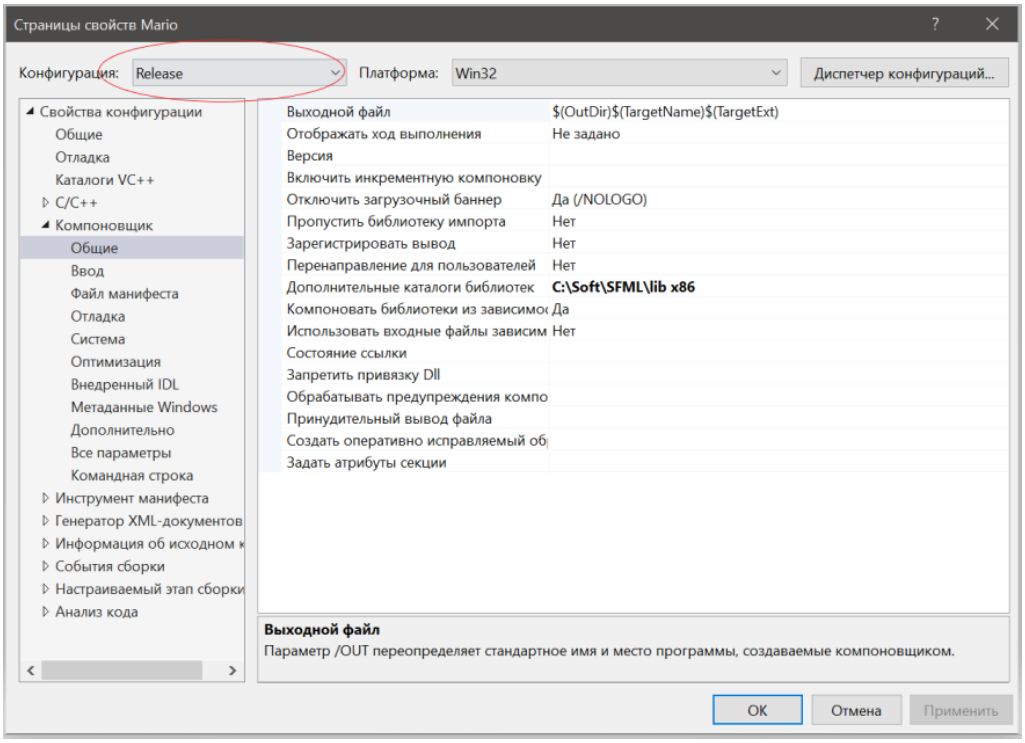

Рис. 3.8 Realese.

После всех этих действий запускаем проект через Локальный отладчик Windows (Рис. 3.9).

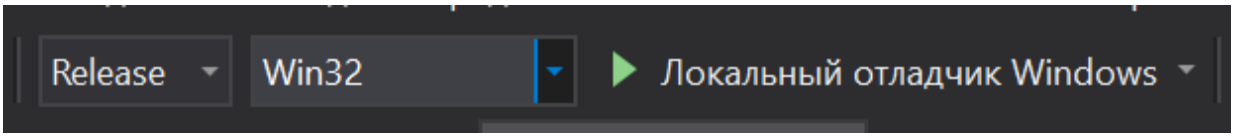

Рис. 3.9 Запуск.

3.1.4 Завершение программы

Для завершения работы программы, если она запущена и вы еще находитесь в “Меню”, можно нажать на кнопку “Exit”, или на красный крестик в правом верхнем углу, если вы уже попали в игру, также через красный крестик.

Заключение

В результате выполнения курсового проекта была написана игра “Mario” для развлечения и приятного проведения досуга.

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по созданию игр в Microsoft Visual Studio на основе библиотеки SFML.

Так же планируется продолжать работу над данным проектом с целью расширения возможностей и увеличения интереса потенциальных игроков.

To-do лист:

1. Увеличение количества уровней и их проработанности;
2. Добавление новых механик и возможностей игроку;
3. Увеличение количества противников и их вариативности;
4. Создание режима для 2 игроков.

Список литературы и интернет-источников

1. Уроки разработки игр на SFML:
   https://kychka-pc.ru/category/sfml
2. Технология разработки программного обеспечения:
   https://znanium.com/catalog/document?pid=1011120
3. Теоретические материалы:
   https://ru.wikipedia.org/
4. Существующие разработке по теме игры-платформеры:
   http://flashdozor.ru/rubric-21-1.html
5. Оптимизация программного кода:
   https://techrocks.ru/2019/01/25/code-optimization-tips/
   https://xakep.ru/2005/01/14/25216/
   https://tech-geek.ru/program-optimization/

Приложения

Приложение 1. Код главного модуля main.cpp

```
// Подключаем часть библиотеки SFML с графикой
#include <SFML/Graphics.hpp>
// Подключаем часть библиотеки SFML с музыкой
#include <SFML/Audio.hpp>
#include <iostream>
#include <sstream>
#include "map.h"
#include "player.h"
#include "enemy.h"

using namespace sf;

//Переменные смещение по осям X и Y.
float offsetX = 0, offsetY = 0;


int main()
{
  int countJump = 0;
  // Создаем окно 400x250 с названием.
    RenderWindow window(VideoMode(400, 250), "Mario game");

  Font font; // Шрифт
  // Передаем нашему шрифту файл шрифта
  font.loadFromFile("CyrilicOld.ttf");
  // Создаем объект текст. закидываем в объект
  // текст строку, шрифт, размер шрифта (в пикселях),
  // сам объект текст (не строка)
  Text text("", font, 15);
  // Покрасили текст в красный
  text.setFillColor(Color::Red);
  text.setStyle(Text::Bold); // Жирный текст.

  //Передача картинки с фигурками в текстуру
  Texture tileSet;
  tileSet.loadFromFile("Mario_Tileset.png");


  Player Mario(tileSet);
  Enemy enemy;
  enemy.Set(tileSet,48 * 16, 13 * 16);


  Sprite tile(tileSet);

  SoundBuffer buffer; //Создание переменной класса "Буфер Звука"
```

```
//Создание буфера и загрузка музыки в него из файла
buffer.loadFromFile("Jump.ogg");
//Загрузка буфера в переменную класса "Звук"
Sound sound(buffer);

// Создание текстур для меню
Texture menu_texture1, menu_texture2;
Texture menu_texture3, about_texture;
menu_texture1.loadFromFile("111.png"); // Загрузка картинок в текстуры
menu_texture2.loadFromFile("222.png");
menu_texture3.loadFromFile("333.png");
about_texture.loadFromFile("about.png");
// Загрузка текстур в спрайты
Sprite menu1(menu_texture1), menu2(menu_texture2);
Sprite menu3(menu_texture3), about(about_texture);
// Переменная на которой основана работа и закрытие меню
bool Menu = 1;
int MenuNum = 0; // Номер картинки
// Установка картинки на позицию в окне по координатам
menu1.setPosition(100, 30);
menu2.setPosition(100, 90);
menu3.setPosition(100, 150);


// меню
while (Menu) //Пока значение Menu = 1(True)
{
  //Устанавливаем цвет для картинки (Белый)
  menu1.setColor(Color::White);
  menu2.setColor(Color::White);
  menu3.setColor(Color::White);
  MenuNum = 0;
  window.clear(Color(0, 0, 0)); // Очистка окна

  //При наведении курсора мышки на поле в рамках указанных координат
  if (IntRect(100, 30, 300, 50).contains(Mouse::getPosition(window)))
  {
       menu1.setColor(Color::Yellow); //Пункт меню окрашивается в желтый
       MenuNum = 1; //Указываем номер пункта меню
  }
  if (IntRect(100, 90, 300, 50).contains(Mouse::getPosition(window)))
  {
       menu2.setColor(Color::Yellow);
       MenuNum = 2;
  }
  if (IntRect(100, 150, 300, 50).contains(Mouse::getPosition(window)))
  {
       menu3.setColor(Color::Yellow);
       MenuNum = 3;
  }

  // При нажатии левой кнопкой мыши по полю пункта
  if (Mouse::isButtonPressed(Mouse::Left))
  {
```

```
        // Если указан первый пункт "Новая игра",
        // меню прекращает работу и запускается игра
        if (MenuNum == 1)
          Menu = false;
        // При выборе второго пункта меню "О программе".
        if (MenuNum == 2)
        {
          // Заготавливается картинка с описанием
          window.draw(about);
          window.display(); // Выводится на экран
          // Картинка с описанием не закроется пока
          // не будет нажата кнопка "Escape".
          while (!Keyboard::isKeyPressed(Keyboard::Escape));
        }

        //Выбор пункта 3 "Выход" приводит к выходу из программы.
        if (MenuNum == 3)
        {
          window.close();
          Menu = false;
        }

   }

   //Рисуем 3 надписи.
   window.draw(menu1);
   window.draw(menu2);
   window.draw(menu3);

   window.display(); //Вывод на дисплей
}

  Music music; // Создание переменной класса музыки.
// Загружаем в переменную музыку из файла
  music.openFromFile("Mario_Theme.ogg");
  music.play(); // Вызов проигрывания музыки

// Переменная для работы с временем
Clock clock;

// Основной цикл программы осуществляющий
// работу со временем и закрытием окна
  while (window.isOpen())
  {
   // Задаем прошедшее время в микросекундах
   float time = clock.getElapsedTime().asMicroseconds();
   clock.restart(); // Перезапускаем время

   time = time / 500; // здесь регулируем скорость игры


      Event event;
   // Обработка события закрытия окна.
      while (window.pollEvent(event))
```

```
    {
        if (event.type == Event::Closed)
            window.close();
}

// Условие когда при нажатии клавиши "Влево"
// происходит смещение по оси x на -0.1.
if (Keyboard::isKeyPressed(Keyboard::Left))
     Mario.dx=-0.1;

// При нажатии клавиши "Вправо" происходит
// смещение по оси x на 0.1.
 if (Keyboard::isKeyPressed(Keyboard::Right))
    Mario.dx=0.1;

// При нажатии клавиши "Вверх" происходит переход к условию
 if (Keyboard::isKeyPressed(Keyboard::Up))
    if (Mario.onGround) // Если Марио на земле
    {
      // Происходит перемещение персонажа
      // вверх по координате Y
      Mario.dy = -0.27;
      Mario.onGround = false;
      // Включается проигрывание ранее
      // заготовленного звука прыжка
      sound.play();
      // Увеличивается счетчик прыжков
      countJump += 1;
    }


// Вызов функции обновления с параметрами, класса PLAYER.
 Mario.Update(time, offsetX, offsetY);
 enemy.Update(time, offsetX, offsetY);

 // Условие пересечения прямоугольника модельки
 // "Марио" с прямоугольником модельки "Противника"
 if  (Mario.rect.intersects(enemy.rect))
 {
     if (enemy.life) //Если противник жив
     {
      if (Mario.dy > 0) //И персонаж игрока приземлился, тогда
      {
            enemy.dx = 0; //Прекращается перемещение противника
            Mario.dy = -0.2; //Персонаж игрока подпрыгивает
            enemy.life = false;
      }
      else
      {
            // Если игрок сталкивается с врагом, а не подает
            // на него, то он окрашивается в красны цвет
            Mario.sprite.setColor(Color::Red);
      }
     }
```

```
}

// Если левая координата прямоугольника > 200
if (Mario.rect.left > 200)
     offsetX = Mario.rect.left - 200; // Смещение



// Очищаем окно голубым цветом (Фон)
window.clear(Color(107, 140, 255));

// Цикл прохождения массива карты по ширине и высоте
for (int i=0; i < H; i++)
     for (int j=0; j < W; j++)
      {
       // Если в массиве встречен символ P, то по заданным
       // координатам с картинки, формируется модель текстуры
      if (TileMap[i][j] == 'P')
            tile.setTextureRect(IntRect(143-16 * 3, 112, 16, 16));

      if (TileMap[i][j] == 'k')
            tile.setTextureRect(IntRect(143, 112, 16, 16));

          if (TileMap[i][j] == 'c')
            tile.setTextureRect(IntRect(143 - 16, 112, 16, 16));

      if (TileMap[i][j] == 't')
            tile.setTextureRect(IntRect(0, 47, 32, 95 - 47));

      if (TileMap[i][j] == 'g')
            tile.setTextureRect(IntRect(0, 16 * 9 - 5, 3 * 16, 16 * 2 +
            5));
      if (TileMap[i][j] == 'G')
            tile.setTextureRect(IntRect(145, 222, 222 - 145 , 255 - 222));

        if (TileMap[i][j] == 'd')
            tile.setTextureRect(IntRect(0, 106, 74, 127 - 106));

      if (TileMap[i][j] == 'w')
            tile.setTextureRect(IntRect(99, 224, 140 - 99, 255 - 224));

      if (TileMap[i][j] == 'r')
            tile.setTextureRect(IntRect(143-32, 112, 16, 16));

      if ((TileMap[i][j] == ' ') || (TileMap[i][j] == '0'))
            continue;

      // Текстура устанавливается на тоже место, что и
      // соответствующий символ массива, по заданным размерам.
        tile.setPosition(j * 16 - offsetX, i * 16 - offsetY) ;
       window.draw(tile);
    }

std::ostringstream playerScoreString;
```

```
     // Занесли в переменную число очков, то есть формируем строку
     playerScoreString << countJump;
     // Задаем строку тексту и вызываем
     // сформированную выше строку
     text.setString("Прыжки:" + playerScoreString.str());
     // Задаем позицию текста, отступая от центра камеры
     text.setPosition(0, 0);
     window.draw(text); // рисую этот текст

     window.draw(Mario.sprite); //Рисуем персонажа "Марио"
       window.draw(enemy.sprite); //Рисуем персонажа "Противник"

     window.display(); // Вывод на дисплей
    }

    return 0;
}
```

Приложение 2. Код модуля player.cpp

```
#include "player.h"
#include "map.h"

// Ширина, и высота используемой рамки
const int size = 16;
// Степень уменьшения скорости анимации
const double period = 0.005;

// Конструктор класса PLAYER отвечающий за получение
// и определение картинки в спрайт,
// а также создания рамки и определения смещения
Player::Player(Texture &image)
{
  // Определяем передаваемую ранее текстуру в спрайт.
  sprite.setTexture(image);
  // Задаем прямоугольник, x, y координаты верхней левой
  // координаты прямоугольника, остальные это ширина и высота
  rect = FloatRect(100, 180, size,size);

  dx = dy = 0.1;
  currentFrame = 0;
}

// Эта функция осуществляет синхронизацию со временем,
// реализацию функции коллизии, а также анимации персонажа.
void Player::Update(float time, float offsetX, float offsetY)
   {

    rect.left += dx * time;
     Collision(0);


    if (!onGround)
```

```
      dy += 0.0005 * time; // Скорость (высота) прыжка вверх

    rect.top += dy * time;
    onGround = false;
     Collision(1);

     currentFrame += time * period; //Привязка кадра по времени к программе
    //Условие смены кадров (картинок всего 3, меняется положение ног)
     if (currentFrame > 3)
      currentFrame -= 3;


    if (dx > 0)
     //Осуществление смены кадров при движении персонажа вперед
     sprite.setTextureRect(IntRect(112 + 31 * int(currentFrame), 144, size,
     size));
   if (dx < 0)
     //Осуществление смены кадров при движении персонажа
     //назад (определено инверсией от движения вперед)
     sprite.setTextureRect(IntRect(112 + 31 * int(currentFrame) +size, 144,
     size * -1, size));

    //Определение позиции появления персонажа
    sprite.setPosition(rect.left - offsetX, rect.top - offsetY);

    dx = 0;
   }

// Функция осуществляет проверку столкновения персонажа
// с различными объектами на карте, играющими роль барьеров
void Player::Collision(int num)
{


  for (int i = rect.top / size; i < (rect.top + rect.height) / size; i++)
    for (int j = rect.left / size; j < (rect.left + rect.width) / size; j++)
    {
         //Условие нахождения различных объектов на карте.
         if ((TileMap[i][j] == 'P') || (TileMap[i][j] == 'k') || (TileMap[i][j]
         == '0') || (TileMap[i][j] == 'r') || (TileMap[i][j] == 't') ||
         (TileMap[i][j] == 'c'))
         {
           // Определение верхней координаты прямоугольника, при
           // условии нахождения персонажа в воздухе и движения вниз.
           if (dy > 0 && num == 1)
           {
                 rect.top = i * size - rect.height;
                 dy = 0;
                 onGround = true;
           }
           // Определение верхней координаты прямоугольника, при
           // условии нахождения персонажа в воздухе и движения вверх.
           if (dy < 0 && num == 1)
           {
```

```
                    rect.top = i * size + size;
                    dy = 0;
                }
                // Определение левой координаты прямоугольника, при
                // условии движения персонажа вперед и по земле.
                if (dx > 0 && num == 0)
                {
                    rect.left = j * size - rect.width;
                }
                // Определение левой координаты прямоугольника, при
                // условии движения персонажа назад и по земле.
                if (dx < 0 && num == 0)
                {
                    rect.left = j * size + size;
                }
            }
        }
}
```

Приложение 3. Код модуля enemy.cpp

```
#include "enemy.h"
#include "map.h"

// Ширина, и высота используемой рамки
const int size = 16;
// Степень уменьшения скорости анимации
const double period = 0.005;

// Функция создает спрайт персонажа по передаваемой
// текстуре и создает прямоугольник (рамку)
void Enemy::Set(Texture &image, int x, int y)
{
  sprite.setTexture(image);
  rect = FloatRect(x, y, size, size);

  dx = 0.05;
  currentFrame = 0;
  life = true;
}

// Эта функция осуществляет синхронизацию со временем,
// реализацию функции коллизии, а также реализует анимацию персонажа.
void Enemy::Update(float time, float offsetX, float offsetY)
{
  rect.left += dx * time;

  Collision();


  currentFrame += time * period;
  if (currentFrame > 2)
    currentFrame -= 2;
```

```
  // Определяет анимацию противника.
  sprite.setTextureRect(IntRect(18 * int(currentFrame), 0, size, size));
  // Определяет анимацию противника после победы героя над ним.
  if (!life) sprite.setTextureRect(IntRect(58, 0, size, size));

  // Установка спрайта противника по заданным координатам.
  sprite.setPosition(rect.left - offsetX, rect.top - offsetY);

}

// Функция проверки столкновения противника с трубами
// и нахождения на земле.
void Enemy::Collision()
{
  // Цил проходящий по объектам карты
  for (int i = rect.top / size; i < (rect.top + rect.height) / size; i++)
    for (int j = rect.left / size; j < (rect.left + rect.width) / size; j++)
        // Условие нахождения объекта на карте с соответствующими ключами
        if ((TileMap[i][j] == 'P') || (TileMap[i][j] == '0'))
        {
          if (dx > 0) // Условие движения противника вправо.
          {
                // Определение точки столкновения
                rect.left = j * size - rect.width;
                // Осуществляет блуждание персонажа вперед и назад,
                // отталкивающегося от объектов.
                dx *= -1;
          }
          else if (dx < 0) // Условие движения персонажа влево.
          {
                rect.left = j * size + size;
                dx *= -1;
          }
        }
}
```

Приложение 4. Код заголовочного файла player.h

```
#ifndef PLAYER_H
#define PLAYER_H
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <sstream>

using namespace sf;

class Player
{

public:

  float dx, dy; // Переменные сдвига по оси x и y
```

```
  // Переменная для заключения нашей картинки в прямоугольник.
  FloatRect rect;
  // Переменная для определения нахождения персонажа на земле.
  bool onGround;
  // Переменная для объединения текстуры с прямоугольником
  Sprite sprite;
  // Переменная для хранения текущего кадра
  //и отрисовки анимации персонажа
  float currentFrame;

  Player(Texture &image);
  void Update(float time, float offsetX, float offsetY);
  void Collision(int num);
};
#endif // PLAYER_H
```

Приложение 5. Код заголовочного файла enemy.h

```
#ifndef ENEMY_H
#define ENEMY_H
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <sstream>

using namespace sf;

class Enemy
{

public:
  float dx, dy; // Переменные смещения по осям X и Y.
  FloatRect rect; // Переменная для заключения картинки в прямоугольник
  Sprite sprite; // Переменная для объединения текстуры с прямоугольником
  // Переменная хранения в ней текущего кадра и отрисовки анимации персонажа
  float currentFrame;
  bool life; // Переменная определяющая состояние противника.

  void Set(Texture &image, int x, int y);
  void Update(float time, float offsetX, float offsetY);
  void Collision();
};
#endif // ENEMY_H
```

Приложение 6. Код заголовочного файла map.h

```
#ifndef MAP_H
#define MAP_H

#include <SFML/Graphics.hpp>
#define H 17 // Высота карты
#define W 150 // Ширина карты
```

```
using namespace sf;

// Создание карты игры, по средствам заполнения
// массива ключами, с помощью которых в последствии
// будет собрана визуальная реализация карты из картинок.
const String TileMap[H] = {
"00000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000",
"0
0",
"0                                                                            w
0",
"0                    w                                  w                    w
0",
"0                                                                            w
0",
"0
0",
"0                                                                            c
0",
"0                                                                            r
0",
"0                                                                           rr
0",
"0                                                                          rrr
0",
"0                ccccckkkkk                                               rrrr
0",
"0                                         t0                             rrrrr
0",
"0G                                        00               t0           rrrrrr
G                                                                0",
"0            d    g        d              00               00          rrrrrrr
0",
"PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP",
"PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP",
"PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP",
};
#endif // MAP_H
```