

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**   
**«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»**

Факультет Физико-технический

Кафедра Компьютерных технологий (КТ)

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | | КТ |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | Т. В. Ермоленко |
| (подпись) | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. | |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе бакалавра 2 курса

на тему:

|  |
| --- |
| РАЗРАБОТКА 2D ИГРЫ В ЖАНРЕ RPG |

|  |  |  |  |
| --- | --- | --- | --- |
| Автор работы |  | Г.А. Захаренко |  |

подпись

|  |  |  |
| --- | --- | --- |
| Направление | 09.03.01 | Информатика и вычислительная техника |

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н., доцент Шарий Т. В.

подпись

Консультанты по разделам:

Сетевые технологии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. преподаватель А.Е. Гукай

подпись

Нормоконтроль \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. лаборант В.Г. Медведева

подпись

Курсовая работа защищена \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата итоговая оценка комиссия

Подписи членов комиссии: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Донецк  
2017

ГОУ ВПО «Донецкий национальный университет»

кафедра Компьютерных технологий

Утверждаю

Зав. кафедрой КТ

\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

**ЗАДАНИЕ**

на курсовую работу студента 2 курса Захаренко Г.А.

*Тема курсовой работы*: Разработка 2D-игры в жанре RPG

*Краткая постановка задачи*: 1. Изучить и проанализировать предметную область разработки игр с 2d графикой в жанре RPG. 2. Ознакомиться с программными продуктами для создания приложений, выполнить их установку. 3. Разработать техническое задание на создание приложения. 4. Разработать алгоритмы приложения. 5. Разработать средствами C++ и SFML 2D игру, которая будет иметь основные особенности жанра RPG. 8. Протестировать и описать программное обеспечение; 9. Оформить отчёт.

*Исходные данные*: 1. Документация об разработке игр с 2d графикой в жанре RPG. 2. Документация по Visual Studio. 3. Документация по языку программирования C++. 4. Документация по SFML.

*Ожидаемые результаты*: Десктоп 2D-игра в жанре RPG.

*Календарный план работы:*

|  |  |  |
| --- | --- | --- |
| Даты консультаций | Этапы выполнения работы | Отметки о выполнении |
| 30.02.2017 | Постановка задачи и обсуждение литературы | выполнено |
| 05.03.2017 | Предварительное утверждение содержания отчёта | выполнено |
| 13.03.2017 | Утверждение проекта, алгоритмов, методов, технологий | выполнено |
| 26.03.2017 | Ход реализации проекта | выполнено |
| 28.03.2017 | Обсуждения организации тестирования программы | выполнено |
| 03.04.2017 | Демонстрация программного продукта руководителю | выполнено |
| 11.04.2017 | Оформление отчёта | выполнено |
| 17.04.2017 | Предоставление отчёта руководителю | выполнено |

Дата выдачи задания 30.01.2017 года

Студент Г.А. Захаренко

Руководитель Т.В. Шарий

**АННОТАЦИЯ**

Отчёт о курсовой работе: 54 с., 14 рис., 5 табл., 2 приложения,   
6 источников.

Объект исследования – разработка игр.

Предмет исследования – разработка жанровых игр с выделенными особенностями.

Цель работы – разработать игру c 2D графикой и элементами жанра RPG.

Метод исследования – анализ возможностей языка С++ и библиотеки SFML для создания игр.

В курсовой работе была разработана игра c 2D графикой и элементами жанра RPG. Игра позволяет: выбор персонажа перед началом игры, управлять персонажем с помощью клавиши «пробел», а также с помощью стрелок либо клавиш «WASD», считывать уровень, созданный при помощи редактора уровней Tiled, считывать анимацию персонажей с xml-файлов. Данное приложение может быть использовано в сфере развлечений.

Дальнейшее развитие системы связанно с: расширением функционала; добавлением новых уровней, персонажей, монстров; изменением систем прокачки и получения уровней.

C++, SFML, VISUAL STUDIO, DLL, LIB, RPG, 2D, SPRITE, XML, TILESET, TILED.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc486890569)

[1 Анализ предметной области 6](#_Toc486890570)

[1.1 Состояние вопроса 6](#_Toc486890571)

[1.2 Описание предметной области 7](#_Toc486890572)

[1.3 Актуальность и цель работы 9](#_Toc486890573)

[2 Техническое задание 10](#_Toc486890574)

[2.1 Описание области применения и исходных данных приложения 10](#_Toc486890575)

[2.2 Требования к пользовательским интерфейсам 10](#_Toc486890576)

[2.3 Требования к аппаратным и программным интерфейсам 12](#_Toc486890577)

[2.4 Требования к пользователям продукта 13](#_Toc486890578)

[2.5 Функции продукта 13](#_Toc486890579)

[2.6 Ограничения 13](#_Toc486890580)

[2.7 Сценарии использования 14](#_Toc486890581)

[3 Обоснование выбора инструментальных средств для реализации поставленной задачи 15](#_Toc486890582)

[3.1 Язык программирования С++ 15](#_Toc486890583)

[3.2 SFML 16](#_Toc486890584)

[3.3 XML 16](#_Toc486890585)

[3.4 Visual Studio 17](#_Toc486890586)

[3.5 Tiled 18](#_Toc486890587)

[3.6 Sprite Decomposer 19](#_Toc486890588)

[3.7 TinyXML 20](#_Toc486890589)

[4 Разработка ПО 21](#_Toc486890590)

[4.1 Входные и выходные данные 21](#_Toc486890591)

[4.2 Структура приложения 23](#_Toc486890592)

[5 Тестирование программного продукта 30](#_Toc486890593)

[5.1 Аппаратные, системные и программные требования 30](#_Toc486890594)

[5.2 Руководства установки и использования 31](#_Toc486890595)

[5.3 Описание контрольных примеров 33](#_Toc486890596)

[ЗАКЛЮЧЕНИЕ 35](#_Toc486890597)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc486890598)

[ПРИЛОЖЕНИЕ А 37](#_Toc486890599)

[ПРИЛОЖЕНИЕ Б 39](#_Toc486890600)

[Листинг Б.1 – Файл game.hpp 39](#_Toc486890601)

[Листинг Б.2 – Фрагмент файла Player.hpp 48](#_Toc486890602)

[Листинг Б.3 – Файл main.cpp 54](#_Toc486890603)

# ВВЕДЕНИЕ

Игра - непременный спутник развития человечества. На стадии игры выполняется чрезвычайно важные функции. Они использовались для социализации подрастающего поколения, для подготовки к коллективной охоте, для тренировки. Но учебно-тренировочные функции были не главными в древней игровой деятельности; главное поле игры.

Основными проблемами разработки игр в данное время является их сложность. Раньше разработкой вполне успешно занимались программисты, взяв на себя все аспекты разработки от логики игры, визуальной части до сценария и диалогов. На данный момент, для того, чтобы разработать качественную игру необходимо пользоваться специальными готовыми библиотеками, в которых уже реализованы такие аспекты, как отрисовка обьектов и интерфейса, физика и так далее.

В рамках курсовой работы будет рассмотрена библиотека для разработки игр для языка C++ на примере разработки 2D-игры в жанре RPG, из чего можно выделить частные задачи для достижения данной цели:

1. Проанализировать особенности разработки игр с использованием библиотеки SFML;
2. Изучить особенности 2D-игр и жанра RPG;
3. Изучить технологии: SFML, C++, Visual Studio 2015, XML;
4. Разработать структуру программы, разработать функционал программы, соответствующий техническому заданию;
5. Разработать интерфейс и логическую составляющую игры.

# 1 Анализ предметной области

## 1.1 Состояние вопроса

На сегодняшний день, компьютерные ролевые игры (или просто RPG) являются одним из самых популярных жанров в игровой индустрии. Предположительно, первой компьютерной ролевой игрой является игра Dungeon, вышедшая в 1975 году. Она представляла собой текстовую версию и не содержала ни капли графики. Начиная с конца восьмидесятых, жанр ролевых игр прочно укрепился на платформе PC. Такие RPG игры, как Nethack, Bard`s Tale, Might & Magic знакомы многим людям, стоящим у истоков жанра.

Однако с развитием компьютерных технологий текстовые ролевые игры быстро исчезли, уступив место 2D-играм. Так в 1988 году вышла первая 2D-игра Pool of Radiance, которая стала предшественницей известной серии Neverwinter Nights. Также в этом плане следует отметить серию игр Ultima и Might & Magic – все они пережили множество реинкарнаций, а Ultima вообще перекочевала в сегмент онлайн игр.

Сильнейший толчок к развитию RPG игры получили с появлением 3D-технологий. Первые трехмерные ролевые игры появились еще в 1993 году, но настоящий бум произошел в 1998 году, когда на рынке появились такие проекты, как Baldur`s Gate и Final Fantasy. Чуть позже вышла System Shock 2 – первая игра, которая удачно совместила жанры RPG и FPS. С тех пор на рынок регулярно выходят гибридные ролевые проекты, содержащие элементы стратегий в реальном времени, космических симуляторов и даже гонок.

Сегодня рпг игры продолжают активно развиваться. Такие проекты, как Fallout 3, Mass Effect и Dragon Age задали новую планку в развитии ролевых игр. Они отличаются чрезвычайно глубокой проработкой игрового мира, множеством второстепенных заданий и персонажей, обширными территориями для изучения и интересными системами развития героя.

## 1.2 Описание предметной области

Игра состоит минимум из трех основных компонентов: техническая реализация, арт и геймдизайн. Для больших игр (например, ММО) техническая часть делится на клиента (игру, то, что видят игроки и с чем взаимодействуют) и сервер (то, где происходит сама игра, обсчеты физики, путей, логики и многого другого). Для социальных сетей не менее важна виральность (вирусное распространение), без которого игра не будет прибыльной. Если некачественный геймдизайн, то в игру не будут играть. Если плохой арт, то игру даже смотреть не будут. И так далее, для каждой платформы могут быть свои особенности, которые надо знать. Таким образом, каждая из основных компонентов игры должна быть не ниже определенного уровня, который ожидают получить игроки. По мере развития игровой индустрии, планка для каждой компоненты поднимается все выше, что, в свою очередь, поднимает порог вхождения в индустрию для разработчиков.

Среди библиотек и игровых движков, способствующих разработке можно выделить высокоуровневые движки в стиле Unrial Engine или Unity 5 и низкоуровневые, которые просто содержат код, который может повторятся во многих играх (функции взаимодействий обьектов и тд). В рамках курсовой работы будут рассмотрены низкоуровневые библиотеки для полного ознакомления с особенностями разработки. В качестве библиотек для рассмотрения были выбраны: SFML, OpenGL.

Таблица 1 – Характеристика SFML

|  |  |
| --- | --- |
| Назвние | SFML (Simple and Fast Multimedia Library) |
| Доступные платформы | Windows, Linux, Mac OS X, Android & iOS. |
| Языки | C, D, Java, Python, Ruby, OCaml, .Net и Go |
| Достоинства | * Имеет лицензию zlib/png license (Предоставляется как есть, можно использовать в коммерческих целях с указанием использования библиотеки). * Полностью открытый исходный код. * Cуществует ряд стандартных библиотек, предоставляющих дополнительную функциональность/ |
| Недостатки | * Доступен не на всех ЯП, доступных для SDL (библиотеки, которая является родителем SFML); * Версии разрабатываются только для актуальных версий Visual Studio и других IDE; |

Таблица 2 – Характеристика OpenGL

|  |  |
| --- | --- |
| Назвние | OpenGL |
| Доступные платформы | Любые |
| Языки | Платформонезависимый программный интерфейс |
| Достоинства | Достоинством OpenGL является переносимость. Поскольку библиотека является открытой, любая компания, которая хочет иметь поддержку OpenGL на своей платформе, просто покупает лицензию у SGI, а затем делает его реализацию на своей платформе. |
| Недостатки | * плохая поддержка со стороны ATi; * несколько более сложная работа с новыми (не вошедшими в стандарт) возможностями железа; * это лишь 3D API, для работы с мышью, клавиатурой, джойстиком звуком и т. д. приходится применять какой-либо другой API |

## 1.3 Актуальность и цель работы

Все более расширяющийся рынок персональной электроники, а с ним и рынок развлечений – неоспоримые аргументы в пользу осуществления проектов по созданию новых игр, а повсеместное распространение и большая доступность инструментальных средств, легкий доступ к информационным обучающим и справочным материалам позволяют уверенно повышать эффективность процесса разработки, и уменьшение сопутствующих трудозатрат.

Разработка игры – процесс, пройдя который можно ознакомится, как и с общими аспектами разработки приложений (построение приложения, формирования ТЗ, контроль проекта, выбор инструментальных средств и ознакомление с ними) так и со специфическими особенностями разработки игр (написание сценариев, разработка игрового баланса).

Игра - непременный спутник развития человечества. На стадии игры выполняется чрезвычайно важные функции. Они использовались для социализации подрастающего поколения, для подготовки к коллективной охоте, для тренировки.

Для разработки 2D-игры в жанре RPG следует выполнить следующие шаги:

1. Проанализировать особенности разработки игр с использованием библиотеки SFML;
2. Изучить особенности 2D-игр и жанра RPG;
3. Изучить технологии: SFML, C++, Visual Studio 2015, XML;
4. Разработать структуру программы, разработать функционал программы, соответствующий техническому заданию;
5. Разработать интерфейс и логическую составляющую игры.

# 2 Техническое задание

## 2.1 Описание области применения и исходных данных приложения

2D-игра в жанре RPG может быть применена в области развлечений. Большинство геймеров, предпочитающих РПГ, — мужчины (82.46%) в возрасте от 16 до 20 лет (41.11%), обучающиеся в институте/академии/ университете (42.11%), с опытом игры 6-8 лет (24.56%) или более 10 лет (24.56%), играющие более 25 часов в неделю (26.32%).

Приложение не имеет никаких исходных данных, кроме ввода пользователя. Поскольку, игра разрабатывается для ознакомления с gamedev и жанром RPG, то приложение будет не сложным и для управления должно хватать только клавиатуры. Приложение будет использовать следующие клавиши:

* цифровые клавиши (для выбора персонажа в меню);
* пробел (атака);
* клавиши направления или «стрелочки» (управление персонажем)
* «WASD» (альтернатива «стрелочкам», которая больше прижилась у геймеров).

## 2.2 Требования к пользовательским интерфейсам

Интерфейс программы должен быть не особо сложным, приятным на вид и быть интуитивно понятным. Среди элементов интерфейса обязательно должны быть стандартные кнопки закрытия, свертывания и развертывания приложения, из чего следует, что окно приложения должно быть масштабируемое.

Приложение должно содержать два окна: с выбором персонажа для старта игры и окно с игрой, содержащее весь функционал. Первое окно приложения можно реализовать с помощью возможностей консоли. В таком случаем пользователю будет выводится текст приветствия и предложение выбрать персонажа для игры с описанием способностей каждого. После выбора персонажа он появляется в новом окне и начинается игровой процесс. В игре может быть два финала: прохождение игры, либо смерть персонажа. В любом из этих случаев, после закрытия окна с игрой пользователю должно выбивается сообщение с итогом его игры, после чего очередное предложение выбрать персонажа.

Окно с игровым процессом должно иметь возможность его закрытия и изменение размеров изображение при этом должно масштабироваться. Данное окно не имеет никаких сложных элементов интерфейса, а только выводит информацию об игровом процессе. Блок-схема, изображающая интерфейс главного окна показан на рисунке 1.

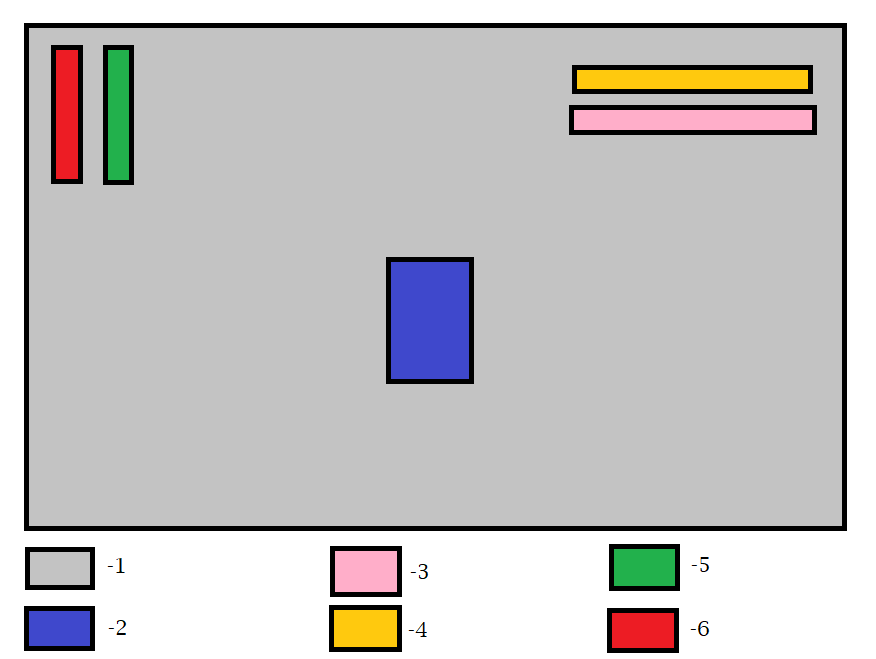


Рисунок 1 – Интерфейс окна с игровым процессом

На рисунке 1 изображены следующие элементы интерфейса:

1. игровое полотно, отображающее карту, все объекты на ней и фон игры;
2. персонаж;
3. количество оставшихся врагов;
4. текущий уровень персонажа;
5. текущий уровень опыта персонажа;
6. текущее количество здоровья персонажа.

Все элементы, кроме первого находятся в статическом положении относительно камеры.

## 2.3 Требования к аппаратным и программным интерфейсам

Для установки и работы программы необходимо иметь вычислительную систему следующей базовой конфигурации:

* процессор: 1.8 ГГц
* оперативная память: от 512 мб
* видеокарта от 256 мб памяти
* более 50 мб памяти на диске для установки

Необходимо обеспечить программное взаимодействие системы с операционными системами Windows XP/7/8/8.1/10. В каталоге с .exe файлом игры должны находиться следующие DLL-файлы для Release сборки проекта:

* sfml-audio-2.dll
* sfml-graphics-2.dll
* sfml-window-2.dll
* sfml-system-2.dll

Для тестовых сборок (Debug) эти библиотеки должны выглядеть следующим образом «sfml-xxxx-d-2.dll».

## 2.4 Требования к пользователям продукта

Пользователи программы должны:

* иметь базовые навыки работы с компьютером

## 2.5 Функции продукта

Пользователи внутри приложения не разделены на классы и всех есть полный доступ к функциям приложения, из которых можно выделить:

* выбор персонажа;
* масштабирование окна игры;
* рестарт игрового процесса с перевыбором персонажа без перезапуска игры.

В ходе игры можно:

* убивать врагов;
* получать опыт;
* повышать уровень;
* производить выстрелы/бить/прыгать на врагов, нанося урон;
* использовать игровые объекты(столбы) как лестницу;
* подыматься по наклонным поверхностям.

## 2.6 Ограничения

* продукт будет поддерживать только русский язык пользовательского интерфейса;
* продукт не предусматривает автоматического перехода на платформы, не перечисленные в данном документе;
* скорость работы приложения будет зависеть только от производительности машины и степени нагрузки ЦП;
* не будет возможности сохранить полученный результат;
* игра не предполагает какие-то настройки игрового процесса или интерфейса

## 2.7 Сценарии использования

Пользователь запускает игру. Открывается консоль с приветствием игрока и предложением выбрать персонажа и кратким описанием возможностей персонажей.

Сценарий 1: пользователь выбирает персонажа. Открывается окно с игровым процессом.

Сценарий 1.1: был выбран персонаж MEGAMAN. Он появился в центре карты. Персонажу доступны умения: передвижение по сторонам, прыжки, передвижение по столбам, стрельба.

Сценарий 1.2: был выбран персонаж МАРИО. Он появился в центре карты. Персонажу доступны умения: передвижение по сторонам, прыжки, передвижение по столбам, удары, прыжки на врага.

Сценарий 1.3: был выбран любой персонаж. Пользователь нажимает на кнопку закрытия окна игры. Окно закрывается фокус пользователя обращается снова в зону консоли.

Сценарий 2: был введенный неверный символ. Выдана ошибка о неверном вводе.

# 3 Обоснование выбора инструментальных средств для реализации поставленной задачи

## 3.1 Язык программирования С++

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

## 3.2 SFML

. SFML (англ. Simple and Fast Multimedia Library — простая и быстрая мультимедиа библиотека) — свободная кроссплатформенная мультимедиа библиотека. Написана на C++, но доступна также для C, D, Java, Python, Ruby, OCaml, .Net и Go. Представляет собой объектно-ориентированный аналог SDL.

SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений. Исходный код библиотеки открытый.

В настоящее время доступны следующие модули:

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

## 3.3 XML

XML (/ˌeks em ˈel/ англ. eXtensible Markup Language) — расширяемый язык разметки. Рекомендован Консорциумом Всемирной паутины (W3C). Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов.

XML удобно использовать в проекте для описание разных обьектов типа уровня или анимации. Для создания вышеупомянутых файлов следует использовать приложения Tiled (создание уровня на основе тайлов) и SpriteDecomposer (создание анимации обьектов в игре на основе спрайт-листов).

Для чтения XML-файлов в игре будет использоваться библиотека TinyXML. Библиотека TinyXML предоставляет программисту очень удобный способ работы с XML. В библиотеке обработаны много вариантов ошибок чтения, а также много методов упрощения доступа к данным, например, получение массива из данных обьектов xml-файла с одинаковым названием и так далее.

## 3.4 Visual Studio

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

## 3.5 Tiled

Tiled — кроссплатформенный открытый редактор тайловых карт для игр. Он позволяет создавать карты для 2-мерных игр (с видом сбоку, таких, как платформеров, или видом сверху), как показано на рисунке 2. Начиная с версии 0.11.0 поддерживаются гексогональные тайлы. Tiled был использован при разработке достаточно большого числа игр, как свободных, так и проприетарных. Сохраняет уровни в основанном на XML формате TMX, чтение которого обеспечивает ряд библиотек для различных языков программирования и такие игровые движки, как Construct 2.

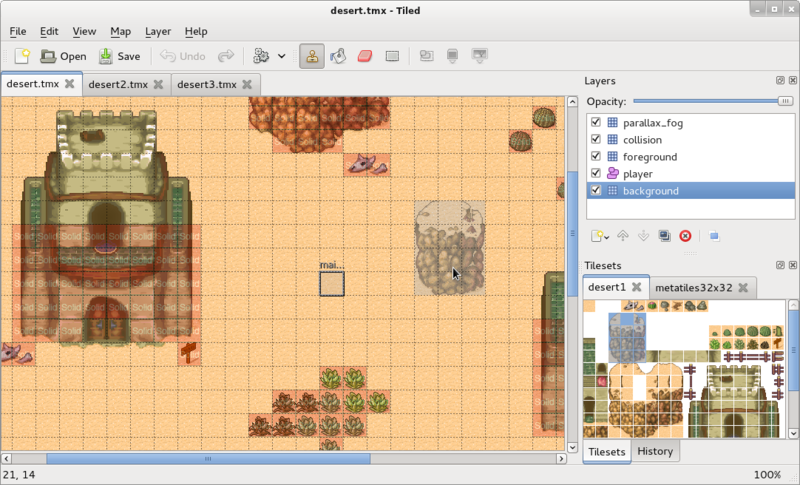


Рисунок 2 – Редактор тайловых карт Tiled

## 3.6 Sprite Decomposer

Вся суть этой простой программы заключается в том, что она открывает так называемые тайлсеты, после чего из отдельных тайлов можно будет собирать анимации. Программа предоставляет разнообразные способы сбора анимации, например, если все тайлы одинаковые по ширине и высоте, и находятся на одинаковом расстоянии друг от друга, то можно выделить их все сразу.

После создания всей анимации ее можно сохранить в .XML файле для дальнейшего хранения. Такой файл можно будет открыть вновь в другой раз и отредактировать, при надобности. Также, удобно то, что сохраненные анимации потом будет легче открыть в программе благодаря известной структуре XML-документа. Пример создания анимации из тайлсета изображен на рисунке 3.

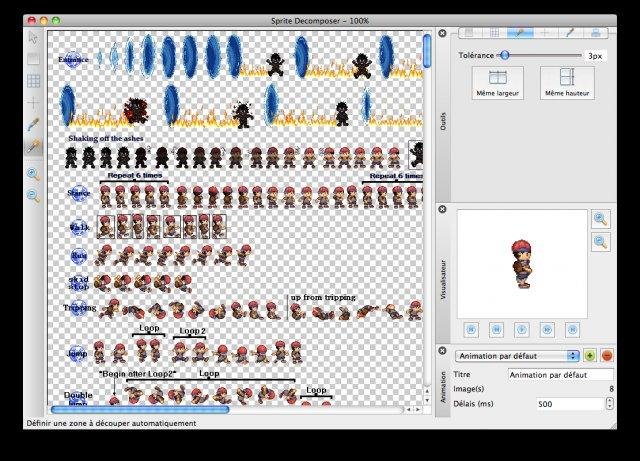


Рисунок 3 – Создание анимации из тайлсета

## 3.7 TinyXML

Для чтения XML-файлов в игре будет использоваться библиотека TinyXML. Библиотека TinyXML предоставляет программисту очень удобный способ работы с XML. В библиотеке много методов упрощения доступа к данным, например, получение массива из данных обьектов xml-файла с одинаковым названием и так далее.

Следует учесть, что файл должен быть валидным, т. е. не содержать ошибок. Например, все теги должны быть правильным образом закрыты (<body /> или <body></body>) и так далее.

# 4 Разработка ПО

## 4.1 Входные и выходные данные

Входные данные приложения изображены в таблице 3.

Таблица 3 – Характеристика OpenGL

|  |  |  |
| --- | --- | --- |
| Данные | Форматы | Описание |
| Тайлсеты | png, jpg | Файл содержащий кадры анимации или шаблоны игрового уровня. |
| Фон | Файл, содержащий изображение, которое отображается за уровнем и его объектами. |
| Карта | tmx | Файл с XML-структурой, в котором находится описание уровня и его обьектов, их позиций, атрибутов и так далее. Данный файл был создан с помощью программы Tiled. |
| Шрифт | ttf | Файл, содержащий шрифт, который используется приложением для отрисовки текста на игровой экран. |
| Динамически подгружаемые библиотеки | dll | Без подключения данных библиотек приложение не сможет использовать необходимые ему функции из библиотеки SFML, так при разработке она подключается динамически. |

Приложение также имеет входные данные в виде ввода пользователя. Приложение использует следующие клавиши:

* цифровые клавиши (1 или 2 для выбора персонажа в меню);
* пробел (атака);
* клавиши направления или «стрелочки» (управление персонажем)
* «WASD» (альтернатива «стрелочкам», которая больше прижилась у геймеров).

Основные выходные данные приложения – это изображение процесса игры, отображения реакции на действие игрока и его влияние на игру. Для этого предусмотрены графические элементы для вывода таких текущих данных:

* шкала здоровья персонажа;
* шкала опыта персонажа;
* уровень персонажа;
* количество оставшихся врагов.

Герой игры отображается в центре экрана, а его движение отображается перемещением мира вокруг него, что позволяет пользователю более углубленно погрузится в мир игры. Состояние игрока характеризуется анимацией. В игре существуют следующие анимации персонажей:

* бег;
* прыжок;
* присед;
* стойка;
* лазанье по лестнице;
* удар/выстрел;

Еще объекты, обладающие анимацией – это патроны и враги. У них по 2 анимации: движение и смерть.

## 4.2 Структура приложения

### 4.2.1 Файловая структура приложения

Файловая структура проекта приложения изображена на рисунке 4.

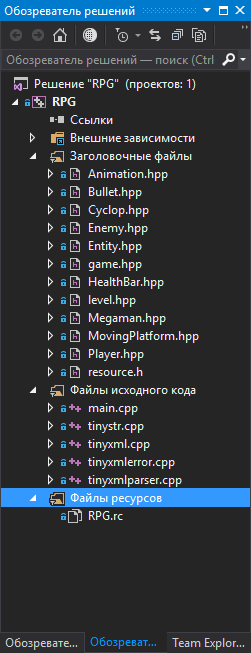


Рисунок 4 - Файловая структура проекта

Рассмотрим детальнее компоненты проекта с приложением (RPG). Он имеет стандартную для С++ структуру проекта и содержит:

* Внешние зависимости (файлы других библиотек и их файлов)
* Ссылки (в данном проекте их нет)
* Заголовочные файлы (файлы с описанием классов их логики)
* Файлы исходного кода
* Фалы ресурсов (используется динамическое подключение)

К заголовочным файлам и файлам исходного кода были подключены также файлы библиотеки TinyXML, так как данный способ подключения данной библиотеки, наиболее оптимальный и рекомендованный производителем.

Также, к файлам проекта можно отнести папку с библиотекой SFML. Она имеет следующие директории:

* Bin – папка с динамически компилируемыми библиотеками SFML
* Doc – папка с документацией
* Examples – папка с примерами работы с библиотекой и ее модулями
* Include – папка со всеми заголовочными файлами библиотеки и ее модулями
* Lib – папка со статическими библиотеками
* cmake

### 4.2.2 Описание обьектов и их взаимодействия

Общая схема взаимодействия обьектов в приложении показана на рисунке 5. Сплошная стрелка символизируют то, что класс, из которого исходит стрелка содержит экземпляр класса, к которому указывает стрелка. В свою очередь пунктирная линия указывает, что тот класс, из которого выходит стрелочка является родительским классом для того, в кого эта стрелка указывает.

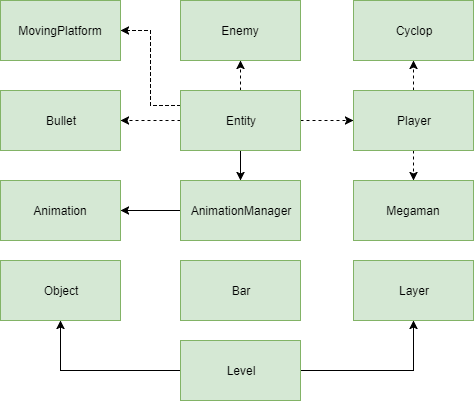


Рисунок 5 - Объекты и их взаимодействие в проекте

Как можно увидеть из рисунка 5, Класс Entity является прародителем почти для всех классов, которые представляют объекты в игре. Это сделано для того, чтобы:

* была возможность не повторять же написанный код;
* для организации полиморфизма.

Если избавление от повторения кода исходит из ТЗ, то полиморфизм нужен игре для того, чтобы можно было организовать такой массив, в котором будут все объекты игры, в свою очередь это позволит организовать один цикл с перебором сразу разных типов обьектов, а не создавать отдельные циклы для каждого сочетания типов, которые взаимодействуют между собой.

Основная информация про классы, содержащиеся на рисунке 5 рассказано в таблице 4.

Таблица 4 – Описание классов разрабатываемого приложения

|  |  |
| --- | --- |
| Название класса | Описание |
| Entity | Класс, который является прародителем для всех классов, которые представляют объекты в игре. Данный класс содержит все методы и атрибуты, которые только могут понадобится объекту во время игрового цикла. Некоторые методы, которые зависят только от самого обьекта, являются виртуальными. |
| MovingPlatform | Данный класс представляет собой обьект движущейся платформы. Единственный метод, который нужен данному классу – это метод update, котоырй он переопределяет от родительского класса Entity. |
| Bullet | Данный класс представляет собой игровой обьект патрона, под которым подразумевается, как и патрон выпущенный персонажем Megaman, так и удар, который производит Cyclop. Единственный метод, который нужен данному классу – это метод update, котоырй он переопределяет от родительского класса Entity. |

Продолжение таблицы 4

|  |  |
| --- | --- |
| Animation | Данный класс отвечает за рисование анимации в игре. Он имеет изображение (тайлсет) и массив координат, в которых находятся кадры анимации, благодаря чему получается анимация. |
| AnimationManager | Класс, который управляет массивом анимации. Каждая анимация прикреплена к «ключу», которым является название анимации, что делает их хранение удобными. В AnimationManager есть полезные методы загрузки анимации из XML-файла, управлением анимации и так далее. |
| Enemy | Данный класс представляет собой игровой обьект врага. Как и большинство подобных классов внутри приложения, данный класс только переопределяет метод update. |
| Player | Данный класс представляет собой игровой обьект персонажа в целом. Как и большинство подобных классов внутри приложения, данный класс переопределяет метод update, но помимо этого данный класс добавил персонажу методы для:   * определения столкновений; * повышения уровня; * изменения состояний анимации в зависимости от действий персонажа; * считывания данных с клавиатуры.   Также, данный класс добавляет атрибуты игроку для возможности реализации регенерации здоровья, а также ограничения персонажа на время между атаками. |

Окончание таблицы 4

|  |  |
| --- | --- |
| Megaman | Данные классы представляют собой игровой обьект конкретного персонажа Megaman или Cyclop. Данный класс наследует класс Player, и контролирует значения характеристик через конструктор (устанавливаются из начальные характеристики) и метод lvlUp(), в котором при повышении уровня характеристики для каждого класса меняются особенным образом (Cyclop получает больше здоровья за уровень, а Megaman – скорости стрельбы). |
| Cyclop |
| Bar | Класс, который позволяет отображать уровень здоровья или опыта в виде элемента, похожего на вертикальный progressbar. |
| Object | Класс, содержащий массив с данными ключ-значение. |
| Layer | Класс, хранящий изображение и уровень его прозрачности. |
| Level | Класс, который отвечает за прорисовку и загрузку всего игрового уровня. Для загрузки используются методы библиотеки TinyXML |

По сути, основным классом является класс Entity, который является базовым классом для всех игровых обьектов. В таблице 5 подробно описаны методы класса Entity.

Таблица 5 – Методы класса Entity

|  |  |
| --- | --- |
| Метод | Описание |
| Entity  (AnimationManager &A,  int X, int Y) | Конструктор класса, вызывает инициализацию компонентов. |
| virtual void update(float time) = 0; | Виртуальный метод, который должен переопределить потомок, чтобы не стать интерфейсом. Логика данного метода в том, что в данном методе должна происходить регуляция всех процессов обьекта (например, движения). Поскольку процессы происходят во времени, то данный метод обязательно должен принимать время (в миллисекундах). |
| void draw(RenderWindow &window) | Данный метод позволяет нарисовать текущее состояние анимации на окно, которое передается в данный метод. |
| FloatRect getRect() | Данный метод возвращает прямоугольник, которое представляет тело обьекта. Данный метод используется в методах, определяющих столкновения с другими объектами. |
| void option  (string NAME,  float SPEED=0,  int HEALTH=10,  string FIRST\_ANIM="") | Метод, реализующий настройку параметров обьекта. |

# 5 Тестирование программного продукта

## 5.1 Аппаратные, системные и программные требования

Аппаратные требования для работы приложения:

* процессор с тактовой частотой 1.0 ГГц;
* оперативная память 512 Мб и более;
* видеокарта с объёмом памяти 64 Мб и выше;
* монитор 800х600 или с более высоким разрешением.

Для функционирования приложения следует, чтобы на компьютере пользователя была установлена операционная система Windows XP или Windows 7/8/8.1/10. Вместе с исполняемым файлом программы должны находится файлы со структурой, изображенной на рисунке 6.

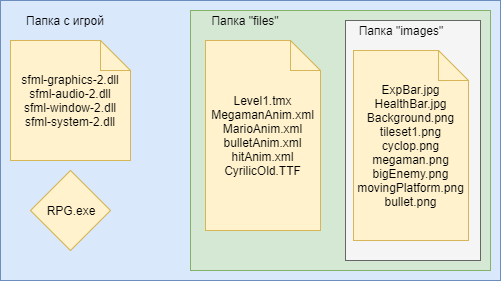


Рисунок 6 – Файловая структура приложения на компьютере пользователя

## 5.2 Руководства установки и использования

### 5.2.1 Руководство пользователю

Для установки приложения на компьютер пользователя необходимо скопировать папку с игрой на жесткий диск пользователя. В случае, если антивирус заблокирует часть файлов или пользователь нарушит файловую структуру, описанную на рисунке 6, то приложение может потерять свою работоспособность.

### 5.2.2 Руководство разработчику

Для разработки приложения и его компонентов необходимо:

* иметь установленную Visual Studio 2015 программы с установленными компонентами для работы c языком C++;
* наличие на компьютере установленного Microsoft Visual C++ Redistributable для Visual Studio 2015;
* установленную на компьютере SFML- 2.4.2 для Visual C++ 14 (2015) - 32-bit

Первое, что необходимо сделать для настройки SFML — это выбрать тип создаваемого проекта: «Win32 application». Мастер предложит вам несколько опций для настройки проекта: выберите «Console application» в том случае, если вам нужна консоль, либо «Windows application» в обратном случае. Выберите «Empty project», если вам не нужен автоматически сгенерированный код.

Создайте файл main.cpp и добавьте его в проект. Этим вы примените настройки C++ (в противном случае Visual Studio не будет знать, какой язык мы будем использовать для данного проекта).

Теперь вам необходимо указать компилятору, где искать заголовочные файлы (файлы с расширением .hpp) и компоновщику, где искать библиотеки SFML (файлы с расширением .lib). Добавьте в свойства проекта следующее:

* Путь до заголовочных файлов SFML (<путь-к-установке-SFML>/include) в C/C++ » General » Additional Include Directories
* Путь до библиотек SFML (<путь-к-установке-SFML>/lib) в Linker » General » Additional Library Directories

Эти пути совпадают для конфигураций Debug и Release, так что вы можете установить их глобально для вашего проекта («All configurations»), как это указанно на рисунке 7.

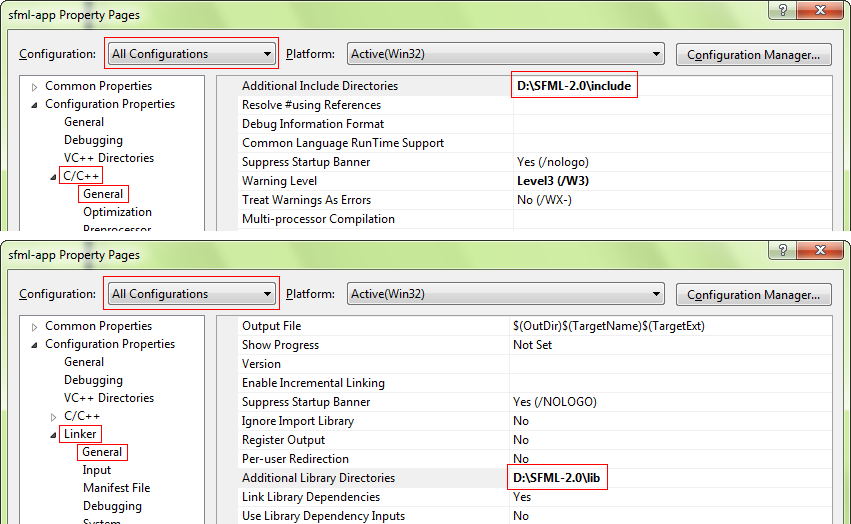


Рисунок 7 – Пример настройки проекта для использования SFML

Следующий шаг — компоновка вашего приложения с библиотеками SFML (файлы с расширением .lib). SFML состоит из пяти модулей (system, window, graphics, network и audio) и библиотек для каждого из них. Библиотеки должны быть добавлены в свойства проекта в Linker » Input » Additional Dependencies. Добавьте те библиотеки SFML, в которых вы нуждаетесь, например «sfml-graphics.lib», «sfml-window.lib» и «sfml-system.lib».

Важно указать библиотеки, соответствующие конфигурации: «sfml-xxx-d.lib» для Debug и «sfml-xxx.lib» для Release, иначе могут возникнуть ошибки.

## 5.3 Описание контрольных примеров

Пользователь запускает приложение, появляется консоль (рисунок 8) с приветствием и предложением выбрать персонажа с описанием каждого.

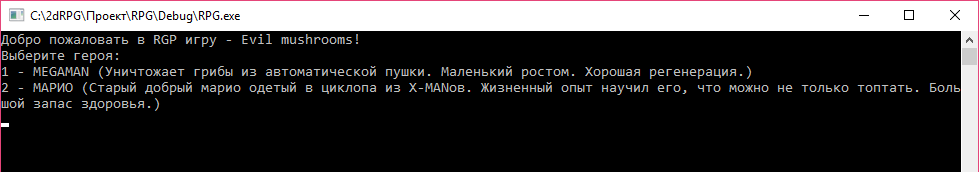


Рисунок 8 – Начальный вид консоли

Пользователь вводит цифру 1, выбирая персонажа MEGAMAN. Открывается окно с игрой, как показано на рисунке 9.

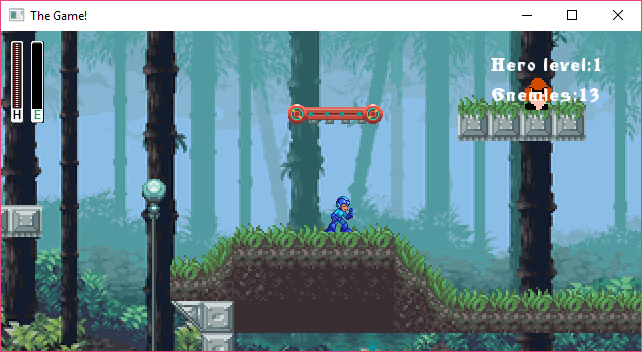


Рисунок 9 – Стартовый вид игры при выборе персонажа MEGAMAN

На рисунке 9 видно, что персонаж находится по центру экрана, находясь в спокойствии. Единственные подвижные объекты на экране –это платформа и гриб, которые двигаются горизонтально и равномерно. При движении персонаж меняет анимацию, при прыжке тоже и так далее. При подходе к подъёму персонаж начинает плавно, равномерно повышаться ближе к врагу. При касании персонажа к врагу персонаж отталкивается на небольшое расстояние назад, изменив при этом анимацию. Со временем шкала здоровья пополняется.

При нажатии на клавишу пробел анимация персонажа изменяется на выстрел и из него вылетает снаряд. При попадании снаряда в гриб снаряд уничтожается. После нескольких попаданий в гриб, последний уничтожается, поменяв анимацию на сплюснутую (рисунок 10.а). Полежав в таком виде три секунды враг исчезает, а шкала опыта персонажа изменяется (рисунок10.б).



Рисунок 10 – Уничтожение врага

а) враг только убит; б) персонаж получил опыт за убийство

При закрытии окна в консоли пишется то, что игрок проиграл и ему снова предлагается сыграть. В этот раз пользователь выбирает второго персонажа. При тех же манипуляциях, персонаж ведет себя почти так же само, разница лишь в атаке. Второй персонаж ударом выпускает снаряд, который летит значительно медленнее, чем у первого, а также пролетает пару секунд и затухает. По мимо этого, при прыжке на врага второй персонаж наносит ему урон и отскакивает от него обратно так, будто еще раз подпрыгнул.

# ЗАКЛЮЧЕНИЕ

В результате работы над курсовой работой выполнены все поставленные задачи. Разработана рабочая 2D-игра в жанр RPG. Был изучен ряд технологий для создания проекта: C++, SFML, TinyXML, Tiled, SpriteDecomposer, Visual Studio 2015. Написано обоснование выбора данных технологий в курсовом проекте. Также были изучены особенности проектирования и реализации игры с графикой 2D в жанре RPG. Функционал приложения был утвержден согласно разработанной структуре программы. На основе разработанной функциональной схемы проекта составлен программный продукт, обладающий интерфейсной частью, работой с файлами в формате XML.

На данный момент при работе с приложением доступны следующие возможности: выбор персонажа перед началом игры, управление персонажем с помощью клавиши «пробел», а также с помощью стрелок либо клавиш «WASD», прохождение игры, преждевременный выход из игры, смерть персонажа, рестарт игрового уровня, считывание уровня созданного при помощи редактора уровней Tiled, чтение анимации персонажей с xml-файлов.

В игре были реализованы следующие особенности: разные боевые механики персонажей, восстановление здоровья со временем, повышение уровня за убийства врагов, возможность использовать лестницу и подъёмы, показ основной информации персонаже и количества оставшихся врагов.

Дальнейшее развитие системы связанно с: расширением функционала; добавлением новых уровней, персонажей, монстров; изменением систем прокачки и получения уровней.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Draw.io – онлайн конструктор диаграмм:

URL: https://www.draw.io/ (дата обращения: 22.03.2017)

2. Professional C++ / Автор: УгатьевИ.,ИвьенБ. / Издательство - М.: Вильямс, Год: 2014.

3. Язык программирования C++. Классика Computers Science: 4-е изд. / М. Торгерсен, С. Вилтамут, П. Голд.

4. [opengl.org](https://www.opengl.org/) - официальный сайт OpenG:

URL: <https://www.opengl.org/about/> (дата обращения: 23.03.2017)

5. SFML – официальный сайт:

URL: https://www.sfml-dev.org/download/sfml/2.4.2/

(дата обращения: 23.03.2017)

URL: https://www.sfml-dev.org/download/learn.php

(дата обращения: 23.03.2017)

6. Wikipedia – информационный ресурс:

URL: https://ru.wikipedia.org/wiki/С++

(дата обращения: 28.03.2017).

URL: https://ru.wikipedia.org/wiki/SFML

(дата обращения: 28.03.2017).

URL: https://ru.wikipedia.org/wiki/XML

(дата обращения: 29.03.2017).

URL: <https://en.wikipedia.org/wiki/List_of_game_engines>

(дата обращения: 01.04.2017).

# ПРИЛОЖЕНИЕ А

**Экранные формы**

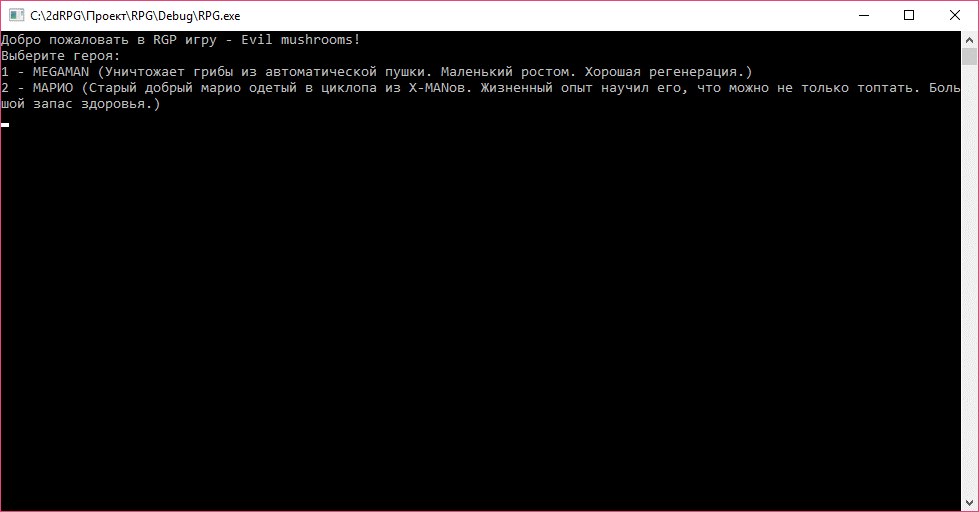


Рисунок А.1 – Начальный вид стартового окна

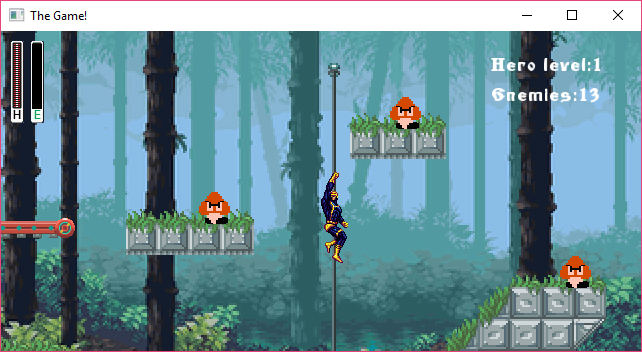


Рисунок А.2 – Изображение гемплея (использование столба как лестницы)

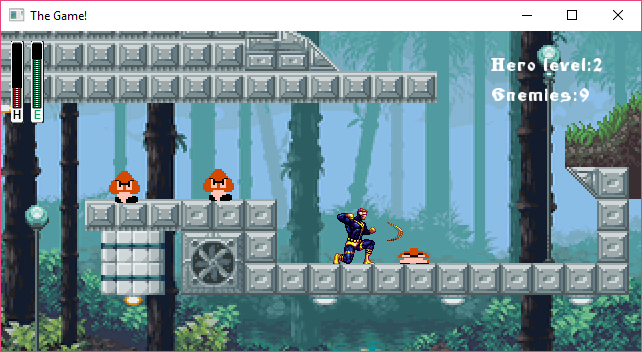


Рисунок А.3 – Использования ближней атаки

для убийства противника

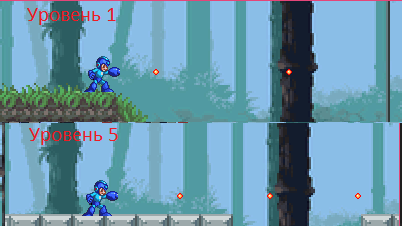
****

Рисунок А.4 Сравнение плотности стрельбы

на разных уровнях прокачки персонажа Megaman

# ПРИЛОЖЕНИЕ Б

**Фрагменты листинга**

## Листинг Б.1 – Файл game.hpp

#ifndef GAME\_H

#define GAME\_H

#include <SFML/Graphics.hpp>

#include <vector>

#include <list>

#include "level.hpp"

#include "Animation.hpp"

#include "Entity.hpp"

#include "Bullet.hpp"

#include "Player.hpp"

#include "Enemy.hpp"

#include "MovingPlatform.hpp"

#include "HealthBar.hpp"

#include "Megaman.hpp"

#include "Cyclop.hpp"

using namespace sf;

using namespace std;

class Game {

public:

};

string RunGame(int heroNumber)

{

///////////// инициализация ///////////////////////////

RenderWindow window(VideoMode(640, 320), "The Game!");

View view(FloatRect(0, 0, 640, 320));

Level lvl;

lvl.LoadFromFile("files/Level1.tmx");

Texture enemy\_t, moveplatform\_t, megaman\_t, bullet\_t, cyclop\_t, bg;

bg.loadFromFile("files/images/bg.png");

enemy\_t.loadFromFile("files/images/bigEnemy.png");

moveplatform\_t.loadFromFile("files/images/movingPlatform.png");

megaman\_t.loadFromFile("files/images/megaman.png");

bullet\_t.loadFromFile("files/images/bullet.png");

cyclop\_t.loadFromFile("files/images/cyclop.png");

AnimationManager megamanAnim;

megamanAnim.loadFromXML("files/anim\_megaman.xml", megaman\_t);

megamanAnim.animList["jump"].loop = 0;

AnimationManager cyclopAnim;

cyclopAnim.loadFromXML("files/anim\_cyclop.xml", cyclop\_t);

cyclopAnim.animList["jump"].loop = 0;

AnimationManager bulletAnim, laserAnim;

bulletAnim.create("move", bullet\_t, 7, 10, 8, 8, 1, 0);

bulletAnim.create("explode", bullet\_t, 27, 7, 18, 18, 4, 0.01, 29, false);

laserAnim.loadFromXML("files/hit.xml", cyclop\_t);

laserAnim.animList["explode"].loop = 0;

AnimationManager anim3;

anim3.loadFromXML("files/enemy.xml", enemy\_t);

anim3.animList["dead"].loop = 0;

AnimationManager anim4;

anim4.create("move", moveplatform\_t, 0, 0, 95, 22, 1, 0);

Sprite background(bg);

background.setOrigin(bg.getSize().x / 2, bg.getSize().y / 2);

background.setScale(2, 2);

std::list<Entity\*> entities;

std::list<Entity\*>::iterator it;

// считывание изначальных врагов

std::vector<Object> e = lvl.GetObjects("enemy");

int enemyCount = e.size();

for (int i = 0; i < e.size(); i++)

entities.push\_back(new ENEMY(anim3, lvl, e[i].rect.left, e[i].rect.top));

e = lvl.GetObjects("MovingPlatform");

for (int i = 0; i < e.size(); i++)

entities.push\_back(new MovingPlatform(anim4, lvl, e[i].rect.left, e[i].rect.top));

Object pl = lvl.GetObject("player");

PLAYER \* playerBuf;

if (heroNumber == 1)

playerBuf = new Megaman(megamanAnim, lvl, pl.rect.left, pl.rect.top);

else

playerBuf = new Cyclop(cyclopAnim, lvl, pl.rect.left, pl.rect.top);

PLAYER & player = \*playerBuf;

Vector2f center = window.getView().getCenter();

Vector2f size = window.getView().getSize();

Bar healthBar("files/images/HealthBar.jpg", Vector2f(center.x - size.x / 2 + 10, center.y - size.y / 2 + 10));

Bar expBar("files/images/ExpBar.jpg", Vector2f(center.x - size.x / 2 + 20, center.y - size.y / 2 + 10));

Font font;

font.loadFromFile("files/CyrilicOld.TTF");

Text lvlLabel("Hero level:", font, 20);

Text enemiesLabel("Enemies:", font, 20);

// set the text style

lvlLabel.setStyle(sf::Text::Bold | sf::Text::Underlined);

enemiesLabel.setStyle(sf::Text::Bold | sf::Text::Underlined);

Clock clock;

/////////////////// основной цикл /////////////////////

while (window.isOpen() && player.life)

{

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

clock.restart();

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

if (time > 40) time = 40;

Event event;

while (window.pollEvent(event))

{

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

window.close();

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

if (event.key.code == Keyboard::Space)

if (player.lastShootTime > player.shoot\_cd) {

if (heroNumber == 1) {

entities.push\_back(new Bullet(bulletAnim, lvl, player.x + 18, player.y + 18, player.dir));

}

else {

entities.push\_back(new Bullet(laserAnim, lvl, player.x + ((player.dir) ? (-10) : (65)), player.y + 25, player.dir, 0.01, 0.5));

}

player.lastShootTime = 0;

}

}

if (Keyboard::isKeyPressed(Keyboard::Left) || Keyboard::isKeyPressed(Keyboard::A)) player.key["L"] = true;

if (Keyboard::isKeyPressed(Keyboard::Right) || Keyboard::isKeyPressed(Keyboard::D)) player.key["R"] = true;

if (Keyboard::isKeyPressed(Keyboard::Up) || Keyboard::isKeyPressed(Keyboard::W)) player.key["Up"] = true;

if (Keyboard::isKeyPressed(Keyboard::Down) || Keyboard::isKeyPressed(Keyboard::S)) player.key["Down"] = true;

if (Keyboard::isKeyPressed(Keyboard::Space)) player.key["Space"] = true;

for (it = entities.begin(); it != entities.end();)

{

Entity \*b = \*it;

b->update(time);

if (b->life == false)

{

if (b->Name == "Enemy") {

player.experience += 50;

enemyCount--;

}

it = entities.erase(it);

delete b;

}

else it++;

}

player.update(time);

for (it = entities.begin(); it != entities.end(); it++)

{

//1. враги

if ((\*it)->Name == "Enemy")

{

Entity \*enemy = \*it;

if (enemy->Health <= 0) continue;

if (player.getRect().intersects(enemy->getRect()))

if (player.dy > 0 && heroNumber == 2) {

player.dy = -0.2;

enemy->Health -= player.damage;

}

else if (!player.hit) {

player.Health -= 33; player.hit = true;

if (player.dir) player.x += 20;

else player.x -= 20;

}

for (std::list<Entity\*>::iterator it2 = entities.begin(); it2 != entities.end(); it2++)

{

Entity \*bullet = \*it2;

if (bullet->Name == "Bullet")

if (bullet->Health > 0)

if (bullet->getRect().intersects(enemy->getRect()))

{

bullet->Health = 0;

enemy->Health -= player.damage;

}

}

}

//2. движущиеся платформы

if ((\*it)->Name == "MovingPlatform")

{

Entity \*movPlat = \*it;

if (player.getRect().intersects(movPlat->getRect()))

if (player.dy > 0)

if (player.y + player.h < movPlat->y + movPlat->h)

{

player.y = movPlat->y - player.h + 3; player.x += movPlat->dx\*time; player.dy = 0; player.STATE = PLAYER::stay;

}

}

//3.. и т.д.

}

/////////////////////отображаем на экран/////////////////////

view.setCenter(player.x, player.y);

window.setView(view);

background.setPosition(view.getCenter());

window.draw(background);

lvl.Draw(window);

for (it = entities.begin(); it != entities.end(); it++)

(\*it)->draw(window);

player.draw(window);

center = view.getCenter();

size = view.getSize();

healthBar.update((player.Health / player.maxHealth),

Vector2f(center.x - size.x / 2, center.y - size.y / 2));

expBar.update((player.experience / float(player.experienceToNextLevel)),

Vector2f(center.x - size.x / 2 + 20, center.y - size.y / 2));

healthBar.draw(window);

expBar.draw(window);

lvlLabel.setString("Hero level:" + to\_string(player.level));

lvlLabel.setPosition(center.x + size.x / 2 - 150,

center.y - size.y / 2 + 20);

window.draw(lvlLabel);

enemiesLabel.setString("Enemies:" + to\_string(enemyCount));

enemiesLabel.setPosition(center.x + size.x / 2 - 150,

center.y - size.y / 2 + 50);

window.draw(enemiesLabel);

window.display();

if (enemyCount==0)

return "Поздравляю, вы прошли игру!";

if (!player.life)

return "Вас убили!";

}

return "Вы не закончили игру!";

}

#endif GAME\_H

## Листинг Б.2 – Фрагмент файла Player.hpp

#ifndef PLAYER\_H

#define PLAYER\_H

#include "Entity.hpp"

class PLAYER : public Entity

{

public:

enum { stay, walk, duck, jump, climb, swim } STATE;

bool onLadder, shoot, hit;

int experience = 0, experienceToNextLevel = 100, level = 1;

float shoot\_cd, damage, hpRegen, lastShootTime = 0;

std::map<std::string, bool> key;

PLAYER(AnimationManager &a, Level &lev, int x, int y) :Entity(a, x, y)

{

option("Player", 0, 100, "stay");

STATE = stay; hit = false;

obj = lev.GetAllObjects();

}

void Keyboard()

{

if (key["L"])

{

dir = 1;

if (STATE != duck) dx = -0.1;

if (STATE == stay) STATE = walk;

}

if (key["R"])

{

dir = 0;

if (STATE != duck) dx = 0.1;

if (STATE == stay) STATE = walk;

}

if (key["Up"])

{

if (onLadder) STATE = climb;

if (STATE == stay || STATE == walk) { dy = -0.27; STATE = jump; anim.play("jump"); }

if (STATE == climb) dy = -0.05;

if (STATE == climb) if (key["L"] || key["R"]) STATE = stay;

}

if (key["Down"])

{

if (STATE == stay || STATE == walk) { STATE = duck; dx = 0; }

if (STATE == climb) dy = 0.05;

}

if (key["Space"])

{

shoot = true;

}

/////////////////////если клавиша отпущена///////////////////////////

if (!(key["R"] || key["L"]))

{

dx = 0;

if (STATE == walk) STATE = stay;

}

if (!(key["Up"] || key["Down"]))

{

if (STATE == climb) dy = 0;

}

if (!key["Down"])

{

if (STATE == duck) { STATE = stay; }

}

if (!key["Space"])

{

shoot = false;

}

key["R"] = key["L"] = key["Up"] = key["Down"] = key["Space"] = false;

}

void Animation(float time)

{

if (STATE == stay) anim.set("stay");

if (STATE == walk) anim.set("walk");

if (STATE == jump) anim.set("jump");

if (STATE == duck) anim.set("duck");

if (STATE == climb) { anim.set("climb"); anim.pause(); if (dy != 0) anim.play(); }

if (shoot) {

anim.set("shoot");

if (STATE == walk) anim.set("shootAndWalk");

}

if (hit) {

timer += time;

if (timer > 1000) { hit = false; timer = 0; }

anim.set("hit");

}

if (dir) anim.flip();

anim.tick(time);

}

void update(float time)

{

Keyboard();

Animation(time);

if (STATE == climb) if (!onLadder) STATE = stay;

if (STATE != climb) dy += 0.0005\*time;

onLadder = false;

if (Health > 0) {

Health += time\*hpRegen / 1000;

if (Health > maxHealth)

Health = maxHealth;

}

else {

life = false;

}

lastShootTime += time / 1000.;

if (experience >= experienceToNextLevel) {

lvlUp();

}

x += dx\*time;

Collision(0);

y += dy\*time;

Collision(1);

}

void lvlUp() {

level += 1;

experience = experience - experienceToNextLevel;

experienceToNextLevel = experienceToNextLevel\*1.1 + 20;

upStats();

}

virtual void upStats() {

}

void Collision(int num)

{

for (int i = 0; i < obj.size(); i++)

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if (dy > 0 && num == 1) { y = obj[i].rect.top - h; dy = 0; STATE = stay; }

if (dy < 0 && num == 1) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (dx > 0 && num == 0) { x = obj[i].rect.left - w; }

if (dx < 0 && num == 0) { x = obj[i].rect.left + obj[i].rect.width; }

}

if (obj[i].name == "ladder") { onLadder = true; if (STATE == climb) x = obj[i].rect.left - 10; }

if (obj[i].name == "SlopeLeft")

{

FloatRect r = obj[i].rect;

int y0 = (x + w / 2 - r.left) \* r.height / r.width + r.top - h;

if (y > y0)

if (x + w / 2 > r.left)

{

y = y0; dy = 0; STATE = stay;

}

}

if (obj[i].name == "SlopeRight")

{

FloatRect r = obj[i].rect;

int y0 = -(x + w / 2 - r.left) \* r.height / r.width + r.top + r.height - h;

if (y > y0)

if (x + w / 2 < r.left + r.width)

{

y = y0; dy = 0; STATE = stay;

}

}

}

}

};

#endif PLAYER\_H

## Листинг Б.3 – Файл main.cpp

#include "source/Game.hpp"

#include <iostream>

using namespace std;

int main()

{

setlocale(0, "rus");

int hero = 0, lvl = 0;

while (true)

{

cout << "Добро пожаловать в RGP игру - Evil mushrooms!" << endl;

cout << "Выберите героя:" << endl;

cout << "1 - MEGAMAN (Уничтожает грибы из автоматической пушки. Маленький ростом. Хорошая регенерация.)" << endl;

cout << "2 - МАРИО (Старый добрый марио одетый в циклопа из X-MANов. Жизненный опыт научил его, что можно не только топтать. Большой запас здоровья.)" << endl;

cin >> hero;

cout << RunGame(hero)<< endl;

}

return 0;

}