СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc500755696)

[1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ ПОСТАВЛЕННОЙ ЗАДАЧИ 7](#_Toc500755697)

[1.1 Обзор языка программирования 7](#_Toc500755698)

[1.2 Обзор среды разработки 7](#_Toc500755699)

[1.3 Обзор библиотеки SFML. сравнение с SDL 8](#_Toc500755700)

[2 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ФАЙЛОВ 10](#_Toc500755701)

[3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА 14](#_Toc500755702)

[4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧ 20](#_Toc500755703)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 23](#_Toc500755704)

[ЗАКЛЮЧЕНИЕ 27](#_Toc500755705)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc500755706)

[ПРИЛОЖЕНИЕ А 29](#_Toc500755707)

[ПРИЛОЖЕНИЕ Б 30](#_Toc500755708)

[ПРИЛОЖЕНИЕ В 31](#_Toc500755709)

[ПРИЛОЖЕНИЕ Г 32](#_Toc500755710)

# ВВЕДЕНИЕ

В настоящее время С++ является языком, наиболее полно представляющим основные парадигмы современного программирования. С++ сочетает в себе три различных принципа программирования:

* Процедурное программирование, представленное языком С и позволяющее создавать библиотеки функций.
* Объектно-ориентированное программирование (ООП), представленное таким понятием как класс и позволяющее разрабатывать библиотеки классов.
* Обобщенное программирование, представленное шаблонами языка С++ и его стандартными библиотеками.

Эти методики программирования делают язык С++ очень мощным, постоянно развивающимся в общем русле эволюции средств программирования и информатики.

Задачей курсовой работы является разработка 2D игры. Эта область программирования очень популярна среди разработчиков, ещё больше она популярна среди простых пользователей, начиная от детей младших классов. Однако индустрия разработки игр наполнена риском. Некоторые из них можно снизить относительно легко, другие наоборот, очень трудно. Рисков много и они разноплановые.

Ультрасовременные компьютерные игры не уступают голливудским блокбастерам по визуальным эффектам, саундтреку и градусу напряжения. Но игры – совершенно иной способ развлечения; хорошая игра может заставить геймера часами просиживать у монитора. Основная изюминка компьютерных игр – интерактивность, которая достигается, прежде всего, с помощью программирования. Именно программа оживляет монстра, эскадрилью штурмовиков или целую армия, заставляет эти виртуальные существа по-разному реагировать на действия пользователя в зависимости от ситуации.

Большинство крупнобюджетных игр (так называемого класса ААА) оснащены своим или заимствованным у коллег «движком». Нередко, впрочем, весь «движок» или его большая часть написана на С++. Именно этот язык использовался при создании множества известных игр – от Doom 3 и Call of Duty до FIFA и The Sims. Впрочем, в освоении С++ есть загвоздка – чрезмерная сложность.

В ходе курсовой работы не использовался «движок», а работа с графическим изображением и реализация взаимодействия игрока с картой и врагами осуществляется при помощи библиотеки SFML (Simple and Fast Multimedia Library). Библиотека SFML представляет простой интерфейс для различных компонент компьютера, чтобы облегчить разработку игр и мультимедийных приложения. Она состоит из пяти модулей: system, window, graphics, audio и network, последние два не использовались в ходе курсовой.

Каждого разработчика привлекает в индустрию разработки игр один важный фактор – высочайшая рентабельность в случае успеха. Просто приведу два последних громких примера – и это не World of Warcraft, нет. Первый – Clash of Clans от финской компании Supercell, которая зарабатывает в сутки $1 млн. Вторая – World of Tanks от Wargaming.net, заработавшая за 2012 год $278 млн. Ещё один плюс – если игра хороша, то разработчик с легкостью сможет найти сильных издателей, которые возьмут на себя все проблемы с локализацией, дистрибуцией и маркетингом игры.

# 1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ ПОСТАВЛЕННОЙ ЗАДАЧИ

## 1.1 Обзор языка программирования

С++ – это компилируемый, статически типизированный язык программи­рования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обоб­щенное программирование. Все языки ООП, включая С++, основаны на трех основополагающих концепциях, называемых инкапсуляцией, полимор­физмом и наследованием. С++ обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. С++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником – языком С, – наибольшее внимание уделено поддержке объектно-ориентированного и обобщенного программирования.

С++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложения для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка С++, 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.

## 1.2 Обзор среды разработки

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 и многое другое.

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 как 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: 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).

## 1.3 Обзор библиотеки SFML. Cравнение с SDL

Библиотека SFML представляет простой интерфейс для различных компонент компьютера, чтобы облегчить разработку игр и мультимедийных приложения. SFML является многоплатформенной библиотекой. С помощью SFML ваше приложение может компилироваться и запускаться в наиболее распространенных операционных системах: Windows, Linux, Mac OS X и вскоре Android и iOS. SFML имеет официальные привязки для языков C и .Net. Благодаря активному сообществу, он также доступен на многих других языках, таких как Java, Ruby, Python, Go и многие другие.

Библиотека SFML состоит из пяти основных модулей: system, window, graphics, audio и network.

1.3.1SFML system – это базовый модуль SFML, определяющий различные утилиты. Он предоставляет векторные классы, строки Unicode и функции преобразования, потоки и мьютексы, классы синхронизации.

1.3.2SFML window предоставляет окна на основе OpenGL и абстракции для событий и обработки ввода.

1.3.3 SFML graphics позволяет визуализировать 2D спрайты, накладывать текст, отображать цвет, рисовать фигуры и т. д.

1.3.4 SFML audio содержит все функции, необходимые для воспроизведения музыки или звуковых эффектов из файла или из буфера.

1.3.5 SFML network – это сотовая связь, утилиты и сетевые протоколы более высокого уровня (HTTP, FTP). Позволяет создавать потоки, блокировать игровые данные, чтобы другие программы знали, что они используются.

SFML не является игровым движком! Это более низкий уровень, и он не включает никаких перетаскиваемых редакторов или средств создания медиа. Это чисто исходный код, и вы, как ожидается, уже знаете, как про-граммировать, чтобы использовать его. Тем не менее, он достаточно мощ-ный, что может помочь вам создать собственный игровой движок.

SFML стремится заменить другую кросс-платформенную библиотеку игр под названием SDL (Simple DirectMedia Layer). SDL намного старше SFML, и был написан и выпущен под GNU LGPL в 1998 году Сэмом Лан-тингом, бывшим ведущим инженером по программному обеспечению Blizzard Entertainment, работающим на World of Warcraft. SDL поддерживает многие игры и до сих пор прекрасно работает, но имеет свои недостатки. Известно, что он довольно загадочен в точках, и поскольку он написан на C, то не использует замечательную объектно-ориентированную парадигму, которая в наши дни так почитается программистами.

SFML был написан на C ++ Лораном Гомилой в 2007 году, лицензирован под лицензией zlib . В отличие от SDL, SFML представляет собой модульную и полностью объектно-ориентированную библиотеку. Его исходный код и SDK очень аккуратные и упорядоченные, что делает его приятным для работы. С точки зрения скорости и производительности SFML либо встречается с SDL, либо иногда превосходит его в некоторых 2D и 3D тестах. В библиотеке много компонентов, но каждый модуль, так сказать, разделен на несколько простых заголовков, которые вы должны включить в свой main.cpp.

Хотя SFML легко настраивается и используется, у него есть недостаток в том, что он имеет меньшую популярность и меньшее сообщество, чем SDL. Из-за этой неясности он поддерживает меньше платформ и имеет меньше учебников и семинаров, чем SDL. Еще раз, это крайне незначительная проблема, так как онлайн-документация SFML, вики и сообщения на форуме – это абсолютно потрясающие справочные ресурсы, и, безусловно, этого вполне достаточно, чтобы исправить любые проблемы, которые могут у вас возникнуть.

# 2 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ФАЙЛОВ

Входные файлы представлены файлами двух видов – файлы формата txt. и файлы формата JPG(PNG).

Входные файлы формата txt. Каждый файл содержит всю необходимую информацию для создания уровня игры. Так как название файлов различается всего на один символ, это позволяет при помощи простого алгоритма отыскать нужный файл и получить доступ к его содержимому (Level1.txt, Level2.txt, …, Level5.txt). Цифра в названии файла соответствует уровню. Как пример рассмотрим структуру файла Level1.txt., представленную на рисунке 2.1.

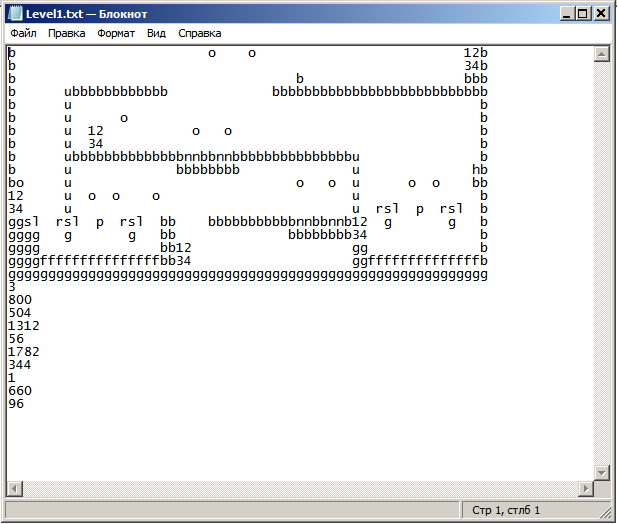


Рисунок 2.1

Структуру файла можно условно разделить на две части: первая – 18 строк длиной 60 символов каждая, вторая – цифры. Рассмотрим каждую из них подробнее:

Первая часть файла предназначена для создания карты уровня (цифра в названии файла говорит о номере уровня). О том, что такое Image, Texture, Sprite , а также подробное описание класса создания карты находится в следующем разделе. PNG файл, предназначенный для визуализации карты, представлен на рисунке 2.2.

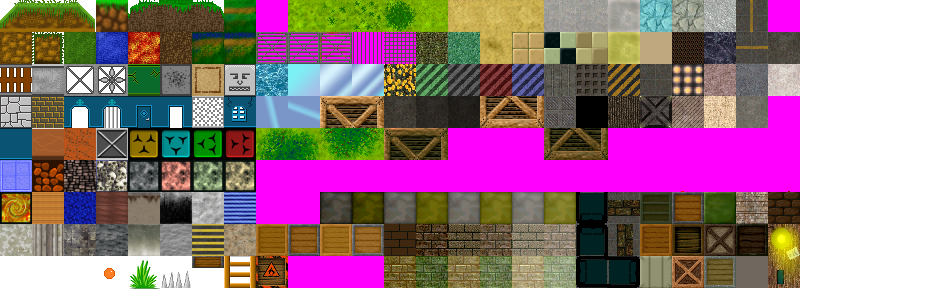


Рисунок 2.2

Вторая часть файла отвечает за количество, а также за местоположение движущихся объектов, с которыми взаимодействует игрок (враги различной сложности, а также движущиеся платформы). Рассмотрим построчно вторую часть файла, но для удобства отобразим её на рисунке 2.3.



Рисунок 2.3

В первой строке содержится информация о количестве врагов на уровне. Так как на нашем рисунке это число 3, значит, следующие 3\*2 строк будут содержать координаты врагов (первое число – координата x, второе число – y). Логика создания врагов такова, что последние две координаты предназначены для «сложного врага», в то время как предшествующие им для «легкого врага». Первая цифра после координат врагов содержит информацию о количестве движущихся платформ, а по аналогии с врагами следующие строки содержат координаты появления платформ. Входные файлы формата PNG, содержащие информацию о врагах и движущейся платформе, представлены на рисунках 2.4 и 2.5 соответственно.

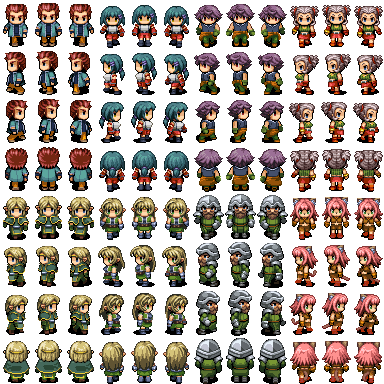


Рисунок 2.4



Рисунок 2.5

Прежде, чем приступить к рассмотрению выходных файлов, следует отметить, что список входных файлов, представленный выше, не полон. Оставшиеся файлы не рассматривались из-за того, что они существуют «самостоятельно», а файлы, рассмотренные нами, существуют «вместе».

Выходной файл представлен бинарным файлом, в котором хранится информация о «лучших» игроках. Содержимое файла RecordTable.dat представляет структуру, изображенную на рисунке 2.6.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Name | Score | Time |
|  | 1 |  | Pavel | 120 | 32 |
|  | .. |  | … | … | … |
|  | 5 |  | Vanya | 80 | 30 |

Рисунок 2.6

Файл RecordTable.dat нельзя назвать «чисто» выходным. Для отображения текущего списка лучших игроков («Top score») производится считывания дынных из файла и вывод их на экран. Описание класса записи информации об лучших игроках в файл представлено в следующем разделе.

# 3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

Код программы можно разделить на несколько блоков: интерфейс приложения, загрузка и обработка данных из файлов, обработка данных в процессе работы приложения и загрузка, обработка и запись данных в файл.

Прежде чем рассмотреть описание каждого блока по отдельности необходимо определить основные принципы создания спрайта, текста, рабочего окна, а также анимации объектов, которые были реализованы в ходе курсовой.

Как говорилось ранее, библиотека SFML содержит модули, один из которых graphics. В данном модуле содержатся классы, при помощи которых создаётся спрайт – Image, Texture, Sprite, текст – Font, Text, рабочее окно – RenderWindow.

Создание спрайта:

1. *Image имя\_изображения; имя\_изображения.loadFromFile(“”)* – *создание объекта класса Image с именем имя\_изображения и загрузка в него файла. В кавычках указывается расположение файла с изображением.*
2. *Texture имя\_текстуры;*

*имя\_текстуры.loadFromeImage (имя\_изображения)* – *создание объекта класса Texture с именем имя\_текстуры.*

1. *Sprite имя\_спрайта; имя\_спрайта.setFromeTexture(имя\_ткстуры)* – *создание объекта класса Sprite с именем имя\_спрайта. Спрайт – это часть текстуры, которая выводится на экран в данный момент времени.*

Создание текста:

1. *Font имя\_шрифта; имя\_шрифта.loadFromFile(“”) – создание объекта класса Font с именем имя\_шрифта. В кавычках указывается расположение файла со шрифтом. Так как в модуле graphics не содержится стандартного шрифта для текста, то необходимо загружать персональные шрифты.*
2. *Text имя\_текста; имя\_текста(“”, размер\_текста, имя\_шрифта)* – *создание объекта класса Text. В кавычках указывается первоначальный текст.*

Создание рабочего окна:

1. *RenderWindow имя\_окна(VideoMode(x, y), “”)* – *создание объекта класса RenderWindow. Первый параметр конструктора с параметрами – VideoMode(x, y) – это размеры нашего будущего окна, второй – в кавычках указывается заголовок нашего рабочего окна.*

Создание анимации объектов:

Анимация – это быстрое переключение спрайтов. Окно SFML зависит от процессора. Если он не загружен – скорость движения героя быстрая, в противном случае – медленная. Чтобы решить эту проблему в ходе курсовой используется «время». Время – бесконечный таймер, который работает, пока открыто окно. Это тот период времени, который проходит между двумя событиями. Задаётся в микросекундах, миллисекундах и секундах. Нам нужна переменная, которая будет передвигать кадры нашей анимации благодаря [времени](http://kychka-pc.ru/sfml/urok-6-sfml-rabota-so-vremenem-obyazatelnaya-chast-sfml.html), она будет хранить в себе текущий кадр. Рассмотрим создание анимации объекта (движение вправо) на примере анимации объекта *player* класса *Player*, которая реализуется в функции *control*:

1. *Player::control(float& time, float& currentFrame) {* – *функция control принимает по ссылке две переменные типа float. Так как игровое время для всех объектов у нас одно, а переменная currentFrame(текущий кадр) зависит от времени, то принимаем по ссылке.*
2. *if(Keyboard::isKeyPressed(Keyboard::right)) {* – *функция control содержит множество проверок, благодаря которым определяется какая анимация сейчас должна осуществляться. Проверка, что нажата клавиша right.*
3. *speed = 0.1;state = right* – *установление направление движения right и скорость игрока равной 0.1.*
4. *currentFrame += 0.0015\*time* – *наращивание переменной currentFrame в зависимости от текущего времени игры;*
5. *if(currentFrame > 3) currentFrame -=3* – *анимация движения вправо реализуется за 3 кадра(1 – правая нога впереди, 2 – ноги вместе, 3 – левая нога впереди). Проверка, если мы дошли до последнего кадра, то возвращаемся к первому;*
6. *sprite.setTextureRect(IntRect(32\*(int)currentFrame), y, width, height)* – *так как спрайт – это часть текстуры, которая выводится на экран в данный момент времени, то с помощью функции setTextureRect выделяется определенный прямоугольник(спрайт(кадр)) из текстуры, где 32\*(int)currentFrame – левый верхний угол спрайта(координата x), y – левый верхний угол(координата y), width – ширина спрайта, height – высота спрайта.*
7. *setPlayerCoordinateForView(x, y)* – *вызывается функция перемещения камеры в зависимости от координат x, y игрока.*

Интерфейс программы реализуется в классе Menu:

1. *private:*
2. *Sprite menu[4], info, menuBackground, name* – *спрайты, которые отображают элементы меню.*
3. *Tecture menuTecture[4], infoTexture, menuBackgroundTexture, nameTexture* – *текстуры, в которых хранятся изображения элементов меню;*
4. *bool isMenu* – *состояние меню.*
5. *int numMenu* – *номер элемента меню.*
6. *void setPosition()* – *метод, который устанавливает позиции элементов меню в окне.*
7. *void setColor()* – *метод, который устанавливает первоначальный цвет элементов меню.*
8. *void draw(RenderWindow& window)* – *метод, который рисует все элементы меню в окне.*
9. *void setNumMenu(RenderWindow& window);* – *метод, который устанавливает значение переменной numMenu в зависимости от выбранного элемента меню, а также меняет цвет элемента при наведении на него курсора.*
10. *public:*
11. *int show(RenderWindow& window)* – *метод, который осуществляет взаимодействие пользователя с меню.*

Загрузка и обработка данных из файла txt. реализуется в классе *Map*, который осуществляет создание карты. В файле проекта map.h объявляются две константные глобальные переменные *const int MAP\_HEIGHT = 18* (высота карты), *const int MAP\_WIDTH = 60*(длина карты):

1. *private:*
2. *static cString \*TileMap* – *массив строк. В каждой строке содержатся символы, каждый из которых означает определенный спрайт карты.*
3. *static int level* – *текущий уровень.*
4. *ifsteam file* – *входной файл.*
5. *char \*buffer* – *буфферная строка для работы.*
6. *Texture textureMap* – *текстура в которой хранится изображение с элементами карты.*
7. *Sprite spriteMap* – *спрайт, который отображает элемент карты, в зависимости от считанного символа.*
8. *public:*
9. *void operator++ (int )* – *метод, который увеличивает на единицу значение текущего уровня и, в зависимости от входного файла, заполняет массив TileMap символами.*
10. *void draw(RenderWindow& window)* – *метод, который, в зависимости от символа массива TileMap, рисует соответствующий ему спрайт в окне.*
11. *static char getTileMapSymbol(int i, int j)* – *метод, который возвращает символ, который соответствует позиции i, j в массиве TileMap.*
12. *static void setTileMapSymbol(int i, int j)* – *метод, который убирает на карте спрайт по позиции i, j.*
13. *static void zero(int i = 0)* – *метод, который устанавливает первый уровень в случае пройгрыша.*
14. *void entitiesCreating(list< Entity\* >& entityList, Image& enemyImage, Image& movingPlatformImage )* – *метод, который, в зависимости от считанных координат из входного файла, добавляет в список врагов и движущиеся платформы.*

Блок обработки данных в процессе работы приложения реализуется в функции *game(Map& Level, RenderWindow& window).* Разобьем текст функции на несколько основных частей: объявление и инициализация, цикл обработки данных. В свою очередь цикл обработки данных разобьем на части: цикл событий, обработка взаимодействий между объектами, графическая часть.

Объявление и инициализация. В ходе курсовой работы была разработана иерархия классов, объекты которой взаимодействуют между собой: объект класса *Player* (персонаж, которым будет играть пользователь); список объектов, с которыми будет взаимодействовать игрок (враги различной сложности, движущиеся платформы, пули врагов). А также объекты класса *Text*, которые будут использоваться в графической части; переменная, отвечающая за «время» и, зависящая от неё, переменная «кадра анимации».

Цикл обработки данных. Этот цикл является обязательным для SFML, который можно перевести для понимания как: «Пока открыто окно – выполняй то, что внутри цикла». Этот цикл бесконечен и выход из него только один – закрыть окно программы.

Цикл событий. Обратите внимание, что мы используем цикл *while* так, чтобы все ожидающие события обрабатывались, если их было несколько. Функция *pollEvent* возвращает true, если событие было отложено, или false, если его не было. Всякий раз, когда мы получаем событие, мы должны проверять его тип (нажатие кнопки «закрыто», нажатие кнопки «мышь») и соответственно реагировать, если мы заинтересованы в этом. Например, если мы заботимся только о событии *Event::Closed*, которое запускается, когда пользователь хочет закрыть окно. На этом этапе окно все еще открыто, и мы должны явно закрыть его с помощью функции закрытия. Это позволяет вам что-то сделать до закрытия окна, например, сохранить текущее состояние приложения или отобразить сообщение.

Обработка взаимодействий между объектами. Эта часть функции осуществляет вызов функций, которые отслеживают корректное взаимодействие игрока с объектами списка, а также взаимодействие всех объектов с картой.

Графическая часть. Во время игры пользователь получает информацию о набранных очках, количестве оставшихся жизней, а также времени, проведенном в игре, благодаря этой части блока.

Загрузка, обработка и запись данных в файл реализуется в классе *Records.* В файле проекта records.h объявляется константная глобальная переменная const int LENGTH= 20. Рассмотрим подробнее переменные и методы класса *Records*:

1. *private:*
2. *static int position* – *смещение записи в таблице.*
3. *static float pos* – *смещение игрока.*
4. *int score* – *очки игрока.*
5. *char playerName[LENGTH]* – *имя игрока.*
6. *int gameTime* – *время, проведенное в игре.*
7. *public:*
8. *void setPlayerName(String name)* – *метод, которые устанавливает значение переменной playerName равной name.*
9. *String getPlayerName() const* – *метод, который возвращает значение переменной playerName.*
10. *void setScore(int number)* – *метод, которые устанавливает значение переменной score равной number.*
11. *int getScore() const* – *метод, который возвращает значение переменной score.*
12. *void setGameTime(int time)* – *метод, которые устанавливает значение переменной gameTime равной time.*
13. *int getGameTime() const* – *метод, который возвращает значение переменной gameTime.*
14. *static void showRecords(RenderWindow& window)* – *описание этого метода представляется в разделе 4.*
15. *static void printTable(RenderWindow& window, Text& buffer)* – *метод, который отображает в окне шапку будущей таблицы результатов.*
16. *static void printPlayers(RenderWindow& window, Text& buffer, Records& player)* – *метод, который выводит в окне информацию, считанную из файла в виде таблицы.*
17. *static void setResult(String playerName, int score, int gameTime)* – *метод, который определяет позицию игрока в «топе» в зависимости от набранных очков.*
18. *static void function(int number, Records& player)* – *описание этого метода представляется в разделе 4.*
19. *static void init()* – *метод, который устанавливает значение переменной pos в зависимости от смещения игрока.*

# 4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧ

Одним из самых объемных и наиболее значимых алгоритмов является функция *entitiesCollision(list<Entity \*>& entityList, list<Entity \*>::iterator& entityIterator, float time)* класса *Player.* Функция принимает список, содержащий объекты, с которыми может взаимодействовать игрок, итератор этого списка, а также переменную, в которой хранится значение времени в данный момент. Рассмотрим этот алгоритм по шагам. (Таблица 4.1)

Таблица *4.1*

|  |  |
| --- | --- |
| № шага | Описание действий |
| 1 | Начало |
| 2 | Объявление переменной (итератор) |
| 3 | Запуск цикла прохода по всем элементам списка |
| 4 | Проверка на пересечение спрайта игрока со спрайтом элемента списка. Если пересекается, то на следующий шаг, иначе на 17 |
| 5 | Проверка на имя элемента списка. Если «MovingPlatform», то на следующий шаг, иначе на 10 |
| 6 | Проверка на состояние игрока. Если падает, то на следующий шаг, иначе на 10 |
| 7 | Проверка на положение игрока. Если находится выше элемента списка, то на следующий шаг, иначе на 10 |
| 8 | Установление координаты «y» игрока таким образом, чтобы он стоял на платформе. |
| 9 | Изменение координаты «x» игрока в зависимости от положения платформы. |
| 10 | Проверка на имя элемента списка. Если «EasyEnemy» или «HardEnemy», то на следующий шаг, иначе на 14 |
| 11 | Проверка на состояние игрока. Если падает, то на следующий шаг, иначе на 13 |
| 12 | Остановка элемента списка. Подбрасывание игрока вверх. Установление переменной здоровья элемента списка равной нулю |
| 13 | Отнятие части переменной здоровья игрока |
| 14 | Проверка на имя элемента списка. Если «EnemyBullet», то на следующий шаг, иначе на 17 |
| 15 | Отнятие части переменной здоровья игрока |
| 16 | Установление переменной здоровья элемента списка равной нулю |
| 17 | Проверка на имя элемента списка. Если «EasyEnemy» или «HardEnemy», то на следующий шаг, иначе на 26 |
| 18 | Инициализация итератора элементом списка из предыдущего шага |

Продолжение таблицы *4.1*

|  |  |
| --- | --- |
| 19 | Запуск цикла прохода по всем элементам списка |
| 20 | Проверка на пересечение и имя. Если элемент списка пересекается с элементом списка, который соответствует итератору и имя элемента списка, которому соответствует итератор «PlayerBullet», то на следующий шаг, иначе на 24 |
| 21 | Увеличение переменной количества очков игрока. |
| 22 | Установление переменной здоровья элемента списка, которому соответствует итератор равной нулю |
| 23 | Установление переменной здоровья элемента списка равной нулю |
| 24 | Увеличение итератора. |
| 25 | Проверка на список. Если конец, то на следующий шаг, иначе на 20 |
| 26 | Переход к следующему элементу списка |
| 27 | Проверка на список. Если не конец, то на шаг 4, иначе на выход |
| 28 | Выход |

Рассмотрим ещё одну функции. Функция *showRecords (RenderWindow& window)* класса *Records*, которая принимает рабочее окно. Рассмотрим алгоритм по шагам этой функции. (Таблица 4.2)

Таблица *4.2*

|  |  |
| --- | --- |
| № шага | Описание действий |
| 1 | Начало |
| 2 | Объявление переменных (*Records player*) |
| 3 | Чистка окна в цвет окна, отображающего меню игры |
| 4 | Загрузка фонового изображения в переменную |
| 5 | Создание спрайта фонового изображения *scoreBackground* и объект *buffer* класса *Text* по алгоритмам, описанным в разделе 3 |
| 6 | Вывод спрайта фонового изображения в окне |
| 7 | Вызов функции *printTable(window, buffer)*класса*Records*, которая отображает в окне шапку будущей таблицы результатов |
| 8 | Открытие существующего файла |
| 9 | Проверка на открытие. Если открылся, то на следующий шаг, иначе на выход |
| 10 | Считывание информацию о первом игроке из файла |
| 11 | Запуск цикла считывания информации об игроках из файла |
| 12 | Проверка файла. Если не конец файла, то на следующий шаг, иначе на 13 |

Продолжение таблицы *4.2*

|  |  |
| --- | --- |
| 13 | Вызов функции *printPlayers(window, buffer, player)* класса  *Records*, которая выводит в окне информацию, считанную из файла в виде таблицы |
| 14 | Считывание информации об игроке |
| 15 | Переход на шаг 12 |
| 16 | Закрытие файла |
| 17 | Отображение окна |
| 18 | Выход |

Рассмотрим ещё две функции:

*function(int number, Records& player)* класса *Records*, которая изменяет в файле запись по индексу *number* на информацию объекта *player*. (Приложение В) А также operator*++(int)* класса *Menu*, которая инициализирует массив строк данными из файла для последующей работы с ним*.* (Приложение Б)

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Минимальные программно-аппаратные требования:

1. Оперативная система – Windows XP, Windows 7 и выше.
2. Оперативная память – не менее 200 мб.
3. Память на жестком диске – 200 мб.
4. Процессор – Intel Celeron(Intel Pentium) с тактовой частотой на менее 1600 Гц.

Для открытия программы необходимо запустить приложение LordOfTheRings.exe. Далее появится стартовое окно – меню, представленное на рисунке 5.1.



Рисунок 5.1

Чтобы ознакомиться с функционалом приложения, а также правилами игры и особенностями прохождения необходимо навести курсор на пункт меню «About» (при наведении курсора цвет меню изменится) и нажать левую клавишу мыши. После нажатия ЛКМ откроется окно, представленное на рисунке 5.2.

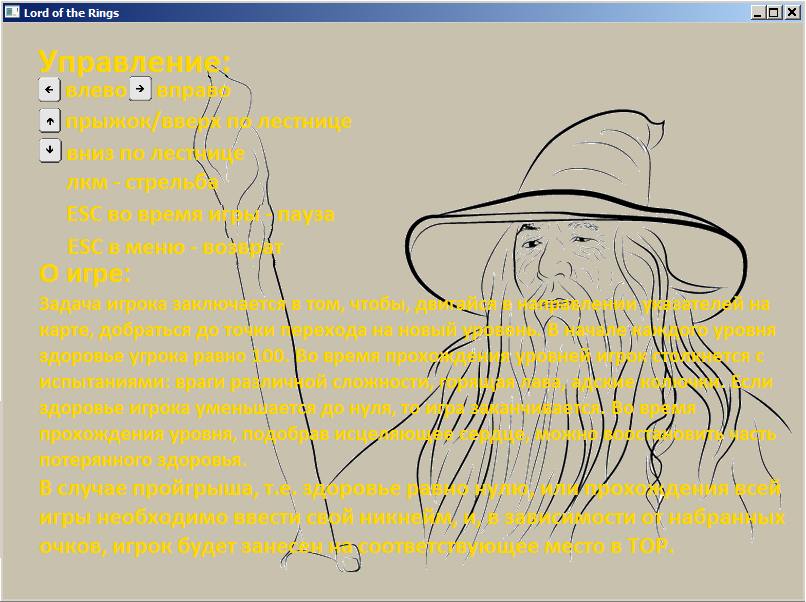


Рисунок 5.2

Чтобы вернуться в главное меню, необходимо, следуя инструкции, нажать клавишу ESC. Чтобы начать новую игру выберите пункт меню «New game». После нажатия ЛКМ откроется окно, представленное на рисунке 5.3.

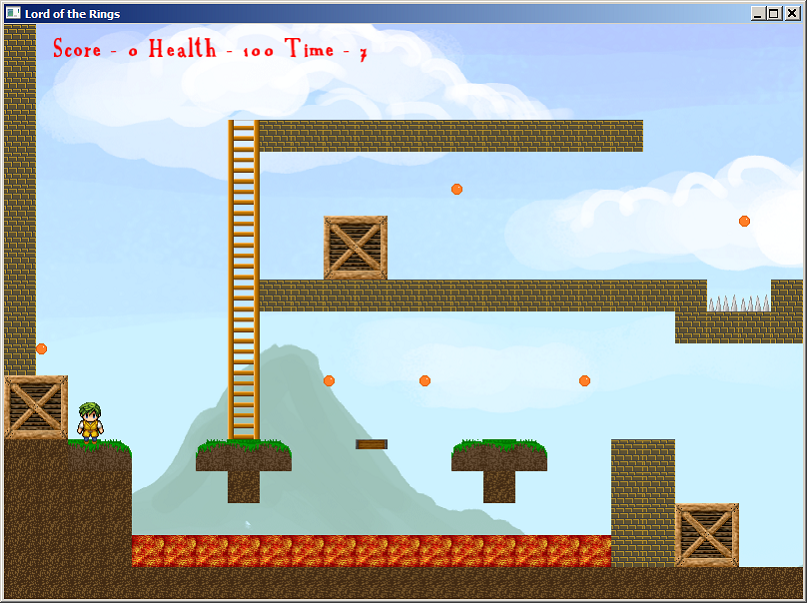


Рисунок 5.3

В случае проигрыша, т.е. здоровье равно нулю, или выигрыша откроется окно, где необходимо ввести «name», для идентификации (окно закроется после нажатия клавиши Enter). Ваш результат будет помещен в «top», если количество набранных вами очков больше либо равно, чем у последнего места. Окно для ввода «name» представлено на рисунке 5.4.

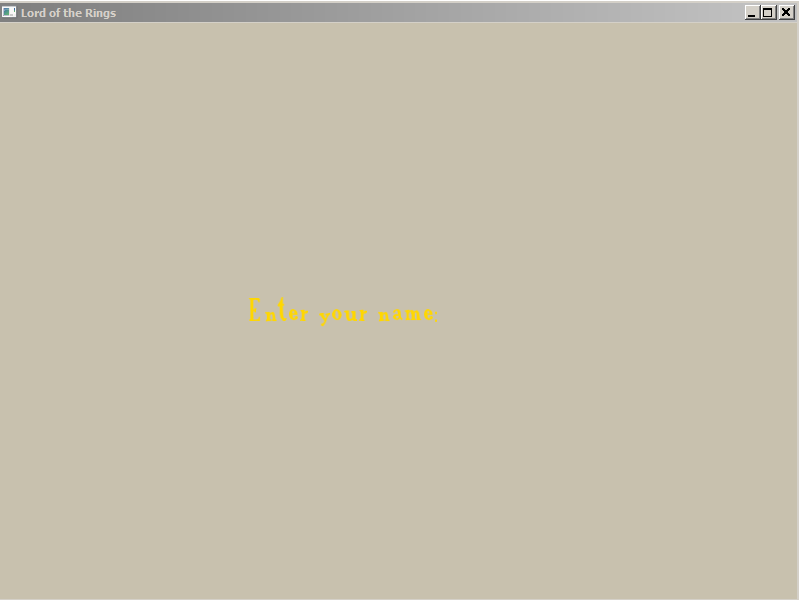


Рисунок 5.4

Чтобы проверить текущий рейтинг игроков необходимо выбрать пункт меню «Top score». После нажатия ЛКМ откроется окно, представленное на рисунке 5.5.



Рисунок 5.5

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано приложение “The Lord Of The Rings”. Приложение не требует большого количества оперативной памяти и времени ожидания, имеет дружественный интерфейс и понятный функционал. При тестировании проекта была проведена корректная обработка ошибок, которые могут возникнуть в ходе эксплуатации приложения.

Таким образом, проект был успешно завершен и протестирован. В ходе разработки были изучены возможности среды разработки Microsoft Visual Studio 2010, получены навыки, которые в будущем станут хорошей основой для создания многоплатформенных игр с использованием библиотеки SFML, на основе различных источников были повышены знания в области ООП и навыки программирования на языке C++.

В ходе выполнения курсовой работы не удалось реализовать корректного и максимально точного взаимодействия игрока с картой, это связано с алгоритмами обработки пересечения игрока со спрайтами карты. Весомой причиной стало отсутствие качественных спрайтов в интернете для создания (прорисовки) персонажей и их анимации.

На этапе разработки была сформулирована идея создания звуковой составляющей игры, которая бы ещё больше погружала в захватывающую атмосферу приложения, однако из-за отсутствия достаточного количества времени от этой идеи пришлось отказаться.

Тематику приложения можно развивать во множестве направлений, одним из которых является поддержка сетевой игры. То есть вы со своим другом можете проходить уровень за уровнем вместе, помогая друг другу в сложных моментах. Так как SFML является многоплатформенной библиотекой, то приложения “The Lord Of The Rings” можно реализовать на мобильных устройствах.

В заключении хочется отметить, что приложение, реализованное в ходе курсовой работы, будет интересно как детям, так и взрослым, ведь интересный сюжет и легкость в управлении всегда привлекают новых пользователей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Луцик, Ю. А. Объектно-ориентированное программирование на языке С++ : учебное пособие / Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова. – Минск: БГУИР, 2003 – 203 с.
2. Дейтел, Х. М. Как программировать на С++ / Х. М. Дейтел, П.Д. Дейтел ; пер. с англ. – М. : Бином, 2007.
3. Standard C++ Library Reference [Электронный ресурс]. – Режим доступа : http://www.cplusplus.com/reference/.
4. SFML и С++ уроки\ Разработка игр [Электронный ресурс]. – Режим доступа : https://www.kychka-pc.ru/.
5. Simple and Fast Multimedia Library [Электронный ресурс]. – Режим доступа : <https://www.sfml-dev.org/>.
6. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон ; пер. с англ. – СПБ. : ДМК, 2004.
7. Доманов, А. Т. Предварительный стандарт предприятия. Дипломные проекты (работы) : общие требования / А. Т. Доманов, Н. И. Сороко. – Минск : БГУИР, 2009.

## ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

## ПРИЛОЖЕНИЕ Б

*(обязательное)*

Блок-схема алгоритма инициализации массива строк данными из файла

## ПРИЛОЖЕНИЕ В

*(обязательное)*

Блок-схема алгоритма изменения информации в файле

## ПРИЛОЖЕНИЕ Г

*(обязательное)*

Листинг кода

//base.h

#ifndef BASE\_H

#define BASE\_H

#include <iostream>

class Base {

protected:

float x, y;

public:

Base() {

x = y = 0;

}

Base(float X, float Y):

x(X), y(Y) {}

virtual float getCoordinateX();

virtual float getCoordinateY();

virtual void setCoordinateX(float X);

virtual void setCoordinateY(float Y);

};

#endif

//base.cpp

#include "base.h"

float Base::getCoordinateX() {

return x;

}

float Base::getCoordinateY() {

return y;

}

void Base::setCoordinateX(float X) {

x = X;

}

void Base::setCoordinateY(float Y) {

y = Y;

}

//entity.h

#ifndef ENTITY\_H

#define ENTITY\_H

#include <iostream>

#include <fstream>

#include <SFML/Graphics.hpp>

#include "base.h"

#include "cstring.h"

using sf::Sprite;

using sf::Image;

using sf::Texture;

using sf::FloatRect;

class Entity: public Base {

protected:

float dx, dy, speed, moveTimer;

int width, height, health;

bool life, isMove, onGround;

Texture texture;

Sprite sprite;

cString name;

public:

Entity(float X, float Y, int WIDTH, int HEIGHT, cString NAME,

Image &image): Base(X, Y), width(WIDTH), height(HEIGHT), dx(0),

dy(0), moveTimer(0), speed(0), health(100),

life(true), onGround(false), isMove(false) {

name = NAME;

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setOrigin(width / 2, height / 2);

}

virtual void checkCollisionWithMap(float, float) = 0;

virtual void update(float&, float&) = 0;

virtual void control(float& time, float& currentFrame) = 0;

virtual Sprite& getSprite();

virtual bool getLife();

virtual char\* getName();

virtual void setDx(float Dx);

virtual void setHealth(int HEALTH);

virtual FloatRect getRect();

virtual double getDx();

virtual int getHeight();

virtual int getWidth();

virtual void setHeight(int HEIGHT);

virtual void setWidth(int WIDTH);

virtual int getHealth();

};

#endif

//entity.cpp

#include "entity.h"

Sprite& Entity::getSprite() {

return sprite;

}

bool Entity::getLife() {

return life;

}

char\* Entity::getName() {

char \*temp = new char[20];

memcpy(temp, name.str(), name.strlen() + 1);

return temp;

}

void Entity::setDx(float Dx) {

dx = Dx;

}

void Entity::setHealth(int HEALTH) {

health = HEALTH;

}

FloatRect Entity::getRect() {

return FloatRect(x, y, width, height);

}

double Entity::getDx() {

return dx;

}

int Entity::getHeight() {

return height;

}

int Entity::getWidth() {

return width;

}

void Entity::setHeight(int HEIGHT) {

height = HEIGHT;

}

void Entity::setWidth(int WIDTH) {

width = WIDTH;

}

int Entity::getHealth() {

return health;

}

//player.h

#ifndef PLAYER\_H

#define PLAYER\_H

#include <iostream>

#include <list>

#include <SFML/Graphics.hpp>

#include "entity.h"

#include "enemy.h"

#include "map.h"

#include "camera.h"

#include "cstring.h"

using sf::Sprite;

using sf::Image;

using std::list;

class Player: public Entity {

static int score;

enum {left, right, up, down, jump, stay} state;

bool onLadder;

public:

bool isShoot;

Player(float X, float Y, int WIDTH, int HEIGHT, cString NAME,

Image &image):

Entity(X, Y, WIDTH, HEIGHT, NAME, image) {

state = stay;

onLadder = false;

if( name == "Player1")

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

}

void checkCollisionWithMap(float, float);

void update(float&, float&);

void control(float&, float&);

void setStartCoordinateXY();

void entitiesCollision(list< Entity\* >& ,

list< Entity\* >::iterator& , float );

void setScore(int value);

int getState();

int getScore();

};

#endif

//player.cpp

#include "player.h"

int Player::score = 0;

void Player::update(float& time, float& currentFrame) {

if(life)

control(time, currentFrame);

switch(state) {

case right: dx = speed; break;

case left: dx = -speed; break;

case up: dy = -speed; break;

case down: dy = speed; break;

case jump: break;

case stay: break;

}

x += dx\*time;

checkCollisionWithMap(dx, 0);

y += dy\*time;

checkCollisionWithMap(0, dy);

if(!isMove) speed = 0;

if(health <= 0) {

setPlayerCoordinateForView(400, 288);

life = false;

health = 0;

speed = 0;

}

sprite.setPosition(x + width / 2, y + height / 2 + 8);

setPlayerCoordinateForView(x, y);

if(!onLadder) {

dy += 0.0015\*time;

}

}

void Player::control(float& time, float& currentFrame) {

if(Keyboard::isKeyPressed(Keyboard::Up) && onLadder) {

speed = 0.1; state = up;

currentFrame += time\*0.0015;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(32\*int(currentFrame), 147, 32, 49));

}

if(Keyboard::isKeyPressed(Keyboard::Up) && onGround && !onLadder) {

dy = -0.5; state = jump; onGround = false;

}

if(Keyboard::isKeyPressed(Keyboard::Down) && onLadder) {

speed = 0.1; state = down;

currentFrame += time\*0.0015;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(32\*int(currentFrame), 147, 32, 49));

}

if(Keyboard::isKeyPressed(Keyboard::Down) && onGround &&!onLadder) {

state = down;

}

if(Keyboard::isKeyPressed(Keyboard::Left)) {

speed = 0.1; state = left;

currentFrame += time\*0.0015;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(32\*int(currentFrame), 49, 32, 49));

setPlayerCoordinateForView(x, y);

}

if(Keyboard::isKeyPressed(Keyboard::Right)) {

speed = 0.1; state = right;

currentFrame += time\*0.0015;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(32\*int(currentFrame), 98, 32, 49));

setPlayerCoordinateForView(x, y);

}

if (Mouse::isButtonPressed(Mouse::Left) && onGround && !onLadder ) {

isShoot = true;

}

onLadder = false;

onGround = false;

}

void Player::checkCollisionWithMap(float Dx, float Dy) {

if(y < 0) y = 0;

int flag = 0;

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

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

if(Map::getTileMapSymbol(i, j) == 'b' ||

Map::getTileMapSymbol(i, j) == 's' || Map::getTileMapSymbol(i, j) == '1' ||

Map::getTileMapSymbol(i, j) == 'g' ||

Map::getTileMapSymbol(i, j) == 'r' ||

Map::getTileMapSymbol(i, j) == '2' ||

Map::getTileMapSymbol(i, j) == 'l' ||

Map::getTileMapSymbol(i, j) == 'p') {

if(Dy > 0) { y = i \* 32 - height; dy = 0;

onGround = true; }

if(Dy < 0) { y = i \* 32 + 32; dy = 0;}

if(Dx > 0) { x = j \* 32 - width; }

if(Dx < 0) { x = j \* 32 + 32; }

}

if(Map::getTileMapSymbol(i, j) == 't') {

health += 20;

Map::setTileMapSymbol(i, j);

}

if(Map::getTileMapSymbol(i, j) == 'u') {

if(Dy > 0 || Dy < 0)

dy = 0;

onLadder = true;

onGround = true;

}

if(Map::getTileMapSymbol(i, j) == 'f'&& Dy > 0 && !flag) {

health -= 40;

dy = -0.6;

onGround = false;

flag++;

}

if(Map::getTileMapSymbol(i, j) == 'o') {

score += 10;

Map::setTileMapSymbol(i, j);

}

}

}

void Player::entitiesCollision(list< Entity\* >& entityList,

list< Entity\* >::iterator& entityIterator, float time) {

list< Entity\* >::iterator it = entityList.begin();

for (entityIterator = entityList.begin();

entityIterator != entityList.end(); entityIterator++) {

if ((\*entityIterator)->getRect().intersects(this->getRect()))

if(!strcmp((\*entityIterator)->getName(),"MovingPlatform")) {

if (dy > 0 || !onGround) {

if (y + height <

(\*entityIterator)->getCoordinateY() +

(\*entityIterator)->getHeight()) {

y = (\*entityIterator)->getCoordinateY()

height + 3;

x += ((\*entityIterator)->getDx()) \* time;

speed = 0; dy = speed;

onGround = true;

continue;

}

}

}

if (!strcmp((\*entityIterator)->getName(), "EasyEnemy") ||

!strcmp((\*entityIterator)->getName(), "HardEnemy")) {

if (dy > 0 && onGround == false) {

(\*entityIterator)->setDx(0);

speed = -0.2; dy = speed;

(\*entityIterator)->setHealth(0);

}

else {

health -= 5;

}

}

if(!strcmp((\*entityIterator)->getName(), "EnemyBullet")) {

health -= 50;

(\*entityIterator)->setHealth(0);

}

}

if (!strcmp((\*entityIterator)->getName(), "EasyEnemy") ||

!strcmp((\*entityIterator)->getName(), "HardEnemy")) {

it = entityIterator;

for(it; it != entityList.end(); it++) {

if((\*entityIterator)->getRect().intersects(

(\*it)->getRect()) &&

!strcmp((\*it)->getName(), "PlayerBullet")) {

this->setScore(40);

(\*entityIterator)->setHealth(0);

(\*it)->setHealth(0);

}

}

}

}

}

void Player::setStartCoordinateXY() {

x = 80;

y = 440;

}

int Player::getState() {

return state;

}

int Player::getScore() {

return score;

}

void Player::setScore(int value) {

this->score += value;

}

//enemy.h

#ifndef ENEMY\_H

#define ENEMY\_H

#include <iostream>

#include <list>

#include <SFML/Graphics.hpp>

#include "entity.h"

#include "map.h"

#include "cstring.h"

using sf::Sprite;

using sf::Image;

class Enemy: public Entity {

public:

Enemy(float X, float Y, int WIDTH, int HEIGHT, cString NAME,

Image &image):

Entity(X, Y, WIDTH, HEIGHT, NAME, image) {

if(name == "EasyEnemy") {

sprite.setTextureRect(sf::IntRect(290, 192, width, height));

dx = 0.1;

}

if(name == "HardEnemy") {

sprite.setTextureRect(sf::IntRect(290, 242, width, height));

}

}

void checkCollisionWithMap(float, float);

void update(float&, float&);

void control(float&, float&);

};

#endif

//enemy.cpp

#include "enemy.h"

void Enemy::checkCollisionWithMap(float Dx, float Dy) {

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

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

if(Map::getTileMapSymbol(i, j) == 'b' ||

Map::getTileMapSymbol(i, j) == 'g' ||

Map::getTileMapSymbol(i, j) == '4') {

if(Dx > 0) {

x = j \* 32 - width; dx = -0.1;

}

if(Dx < 0) {

x = j \* 32 + 32; dx = 0.1;

}

}

}

}

void Enemy::update(float& time, float& currentFrame) {

control(time, currentFrame);

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + width / 2, y + height / 2);

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

}

void Enemy::control(float& time, float& currentFrame) {

if(dx < 0) {

currentFrame += time\*0.003;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(290+32\*int(currentFrame),245,32,40));

}

if(dx > 0) {

currentFrame += time\*0.003;

if(currentFrame > 3) currentFrame -= 3;

sprite.setTextureRect(IntRect(290+32\*int(currentFrame),293,32,40));

}

}

//bullet.h

#ifndef BULLET\_H

#define BULLET\_H

#include <iostream>

#include <SFML/Graphics.hpp>

#include "entity.h"

#include "map.h"

#include "cstring.h"

using sf::Sprite;

using sf::Image;

class Bullet: public Entity {

int direction;

public:

Bullet(float X, float Y, int WIDTH, int HEIGHT, cString NAME,

Image &image, int DIRECTION):

Entity(X, Y, WIDTH, HEIGHT, NAME, image) {

direction = DIRECTION;

speed = 0.8;

life = true;

}

void checkCollisionWithMap(float, float);

void update(float&, float&);

void control(float&, float&) {

return;

}

};

#endif

//bullet.cpp

#include "bullet.h"

void Bullet::update(float& time, float& currentFrame) {

switch (direction) {

case 0: dx = -speed; break;

case 1: dx = speed; break;

default: dx = speed; break;

}

x += dx\*time;

if(x <= 0) life = false;

if(x >= 1920) life = false;

if(health <= 0) life = false;

checkCollisionWithMap(time, currentFrame);

sprite.setPosition(x + width, y + height + 10);

}

void Bullet::checkCollisionWithMap(float , float ) {

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

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

if(Map::getTileMapSymbol(i, j) == 'b' ) {

life = false;

}

}

//movingPlatform.h

#ifndef MOVINGPLATFORM\_H

#define MOVINGPLATFORM\_H

#include <iostream>

#include <SFML/Graphics.hpp>

#include "entity.h"

#include "map.h"

#include "cstring.h"

using sf::Image;

using sf::Sprite;

class MovingPlatform: public Entity {

public:

MovingPlatform(float X, float Y, int WIDTH, int HEIGHT, cString NAME,

Image &image):

Entity(X, Y, WIDTH, HEIGHT, NAME, image) {

sprite.setTextureRect(sf::IntRect(0, 0, width, height));

dx = 0.08;

}

void checkCollisionWithMap(float, float);

void update(float&, float&);

void control(float&, float&);

};

#endif

//movingPlayform.cpp

#include "movingPlatform.h"

void MovingPlatform::update(float& time, float& currentFrame) {

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + width / 2, y + height / 2);

}

void MovingPlatform::control(float& , float&) {

return;

}

void MovingPlatform::checkCollisionWithMap(float Dx, float Dy) {

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

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

if(Map::getTileMapSymbol(i, j) == 'b' ||

Map::getTileMapSymbol(i, j) == 'g' ||

Map::getTileMapSymbol(i, j) == '4') {

if(Dx > 0) {

x = j \* 32 - width; dx = -0.1;

}

if(Dx < 0) {

x = j \* 32 + 32; dx = 0.1;

}

}

}

}

//map.h

#ifndef MAP\_H

#define MAP\_H

#include <iostream>

#include <fstream>

#include <list>

#include <SFML/Graphics.hpp>

#include "base.h"

#include "entity.h"

#include "enemy.h"

#include "movingPlatform.h"

#include "cstring.h"

using namespace sf;

using std::list;

const int MAP\_HEIGHT = 18;

const int MAP\_WIDTH = 60;

class Map {

static cString \*TileMap;

static int level;

ifstream file;

char \*buffer;

Texture textureMap;

Sprite spriteMap;

public:

Map() {}

Map(Image &image) {

buffer = new char[MAP\_WIDTH + 1]; memcpy(buffer, "", MAP\_WIDTH+1);

TileMap = new cString[MAP\_HEIGHT]();

textureMap.loadFromImage(image);

spriteMap.setTexture(textureMap);

}

void operator++ (int);

void draw(RenderWindow &);

static char getTileMapSymbol(int i, int j);

static void setTileMapSymbol(int i, int j);

static void zero(int = 0);

void entitiesCreating(list< Entity\* >& entityList, Image& enemyImage,

Image& movingPlatformImage);

~Map() {

file.close();

delete [] buffer;

delete [] TileMap;

}

};

#endif

//map.cpp

#include "map.h"

cString\* Map::TileMap;

int Map::level = 0;

void Map::operator++ (int) {

level++;

memcpy(buffer, (string("Level") + to\_string((long long int)level) +

string(".txt")).c\_str(), 11);

file.open(buffer, ios::in);

if(!file) exit(1);

int i = 0;

while(i < MAP\_HEIGHT) {

file.getline(buffer, MAP\_WIDTH + 1);

TileMap[i] = buffer;

i++;

}

}

void Map::entitiesCreating(list< Entity\* >& entityList, Image& enemyImage,

Image& movingPlatformImage) {

int number\_of\_enemies;

file.getline(buffer, 5); number\_of\_enemies = atoi(buffer);

Base \*array\_of\_enemies = new Base[number\_of\_enemies]();

int i = 0;

while(i < number\_of\_enemies) {

file.getline(buffer, 5);

array\_of\_enemies[i].setCoordinateX(atoi(buffer));

file.getline(buffer, 5);

array\_of\_enemies[i].setCoordinateY(atoi(buffer));

if(i < number\_of\_enemies - 1)

entityList.push\_back(new Enemy

(array\_of\_enemies[i].getCoordinateX(),

array\_of\_enemies[i].getCoordinateY(), 32, 40,

"EasyEnemy", enemyImage));

if(i == number\_of\_enemies - 1)

entityList.push\_back(new Enemy

(array\_of\_enemies[i].getCoordinateX(),

array\_of\_enemies[i].getCoordinateY(), 32, 40,

"HardEnemy", enemyImage));

i++;

}

int number\_of\_movingPlatforms;

file.getline(buffer, 5); number\_of\_movingPlatforms = atoi(buffer);

Base \*array\_of\_movingPlatforms = new Base[number\_of\_movingPlatforms]();

i = 0;

while(i < number\_of\_movingPlatforms) {

file.getline(buffer, 5);

array\_of\_movingPlatforms[i].setCoordinateX(atoi(buffer));

file.getline(buffer, 5);

array\_of\_movingPlatforms[i].setCoordinateY(atoi(buffer));

entityList.push\_back(new MovingPlatform

(array\_of\_movingPlatforms[i].getCoordinateX(),

array\_of\_movingPlatforms[i].getCoordinateY(), 95, 22,

"MovingPlatform", movingPlatformImage));

i++;

}

file.close();

}

void Map::draw(RenderWindow &window) {

for(int i = 0; i < MAP\_HEIGHT; i++)

for(int j = 0; j < MAP\_WIDTH; j++) {

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

spriteMap.setTextureRect(IntRect(0, 256, 0, 0));

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

spriteMap.setTextureRect(IntRect(32, 96, 32, 32));

if(TileMap[i][j] == '1')

spriteMap.setTextureRect(IntRect(320, 96, 32, 32));

if(TileMap[i][j] == '2')

spriteMap.setTextureRect(IntRect(352, 96, 32, 32));

if(TileMap[i][j] == '3')

spriteMap.setTextureRect(IntRect(480, 96, 32, 32));

if(TileMap[i][j] == '4')

spriteMap.setTextureRect(IntRect(512, 96, 32, 32));

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

spriteMap.setTextureRect(IntRect(192, 0, 32, 32));

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

spriteMap.setTextureRect(IntRect(128, 0, 32, 32));

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

spriteMap.setTextureRect(IntRect(160, 32, 32, 32));

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

spriteMap.setTextureRect(IntRect(160, 0, 32, 32));

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

spriteMap.setTextureRect(IntRect(128, 32, 32, 32));

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

spriteMap.setTextureRect(IntRect(224, 256, 32, 32));

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

spriteMap.setTextureRect(IntRect(128, 256, 32, 32));

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

spriteMap.setTextureRect(IntRect(192, 256, 32, 10));

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

spriteMap.setTextureRect(IntRect(160, 256, 32, 32));

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

spriteMap.setTextureRect(IntRect(104, 268, 11, 11));

spriteMap.setPosition(j \* 32, i \*32);

window.draw(spriteMap);

}

}

void Map::zero(int value) {

level = value;

}

char Map::getTileMapSymbol(int i, int j) {

return TileMap[i][j];

}

void Map::setTileMapSymbol(int i, int j) {

TileMap[i][j] = ' ';

}

//records.h

#ifndef RECORDS\_H

#define RECORDS\_H

#include <iostream>

#include <fstream>

#include <SFML/Graphics.hpp>

#include "cstring.h"

#include "camera.h"

using namespace sf;

using std::string;

const int LENGTH = 20;

class Records {

static int position;

static float pos;

int score;

char playerName[LENGTH];

int gameTime;

//time\_t date\_of\_game;

public:

Records(int = 0, String = "", int = 0);

void setPlayerName(String name);

String getPlayerName() const;

void setScore(int number);

int getScore() const;

void setGameTime(int time);

int getGameTime() const;

static void showRecords(RenderWindow& window);

static void printTable(RenderWindow& window, Text& buffer);

static void printPlayers(RenderWindow& window, Text& buffer, Records& player);

static void setResult(String playerName, int score, int gameTime);

static void function(int number, Records& player);

static void init();

};

#endif

//records.cpp

#include "records.h"

int Records::position = 50;

float Records::pos;

void Records::init() {

pos = view.getCenter().x - 400;

}

Records::Records(int scoreValue, String playerNameValue, int gameTimeValue) {

setPlayerName(playerNameValue);

setScore(scoreValue);

setGameTime(gameTimeValue);

}

int Records::getScore() const {

return score;

}

int Records::getGameTime() const {

return gameTime;

}

String Records::getPlayerName() const {

return playerName;

}

void Records::setScore(int scoreValue) {

score = scoreValue;

}

void Records::setGameTime(int gameTimeValue) {

gameTime = gameTimeValue;

}

void Records::setPlayerName(String playerNameValue) {

string buf = playerNameValue;

const char\* string = buf.c\_str();

int length = playerNameValue.getSize();

length = length < LENGTH ? length : LENGTH - 1;

memcpy(playerName, string, length);

playerName[length] = '\0';

}

void Records::printPlayers(RenderWindow& window, Text& buffer,

Records& player) {

char buf[10];

position += 50;

buffer.setString(player.getPlayerName());

buffer.setPosition(pos + 140, position); window.draw(buffer);

buffer.setString(itoa(player.getScore(), buf,10));

buffer.setPosition(pos + 320, position); window.draw(buffer);

buffer.setString(itoa(player.getGameTime(), buf,10));

buffer.setPosition(pos + 460, position); window.draw(buffer);

if(position == 300) position = 50;

}

void Records::printTable(RenderWindow& window, Text& buffer) {

buffer.setString("Name"); buffer.setPosition(pos + 140, 50);

window.draw(buffer);

buffer.setString("Score"); buffer.setPosition(pos + 300, 50);

window.draw(buffer);

buffer.setString("Time"); buffer.setPosition(pos + 440, 50);

window.draw(buffer);

buffer.setString("Date"); buffer.setPosition(pos + 600, 50);

window.draw(buffer);

}

void Records::showRecords(RenderWindow& window) {

window.clear(Color(200, 193, 174));

Image scoreBackgroungImage;

scoreBackgroungImage.loadFromFile("images/scorebackground.png");

Texture scoreBackgroungTexture;

scoreBackgroungTexture.loadFromImage(scoreBackgroungImage);

Sprite scoreBackgroung;

scoreBackgroung.setTexture(scoreBackgroungTexture);

scoreBackgroung.setPosition(pos + 0, 0);

window.draw(scoreBackgroung);

Font font; font.loadFromFile("Applications/a\_font\_with\_serifs.ttf");

Text buffer("", font, 40); buffer.setColor(Color(255, 215, 0));

buffer.setStyle(Text::Bold);

printTable(window, buffer);

ifstream recordsFile("RecordTable.dat", ios::binary);

Records player;

recordsFile.read(reinterpret\_cast< char\* >(&player), sizeof(Records));

while(recordsFile && !recordsFile.eof()) {

printPlayers(window, buffer, player);

recordsFile.read(reinterpret\_cast< char\* >(&player), sizeof(Records));

}

recordsFile.close();

window.display();

}

void Records::setResult(String playerNameValue, int score, int gameTime) {

init();

Records player(score, playerNameValue, gameTime);

string buffer;

Records object;

int flag = -1, arr[5];

ifstream recordsFile("RecordTable.dat", ios::binary);

if(!recordsFile)

exit(1);

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

recordsFile.read(reinterpret\_cast< char\* >(&object),

sizeof(Records));

arr[i] = object.getScore();

if(object.getPlayerName() == player.getPlayerName()) {

if(object.getScore() <= player.getScore()) {

flag = i;

}

else return;

}

}

recordsFile.close();

if(flag < 0) {

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

if(arr[i] == player.getScore()) {

int j;

for(j = i; arr[j] == arr[i]; j++) ;

function(j, player); break;

}

if(arr[i] < player.getScore()) {

function(i, player); break;

}

}

}

else {

function(flag, player);

}

}

void Records::function(int number, Records& player) {

Records object;

fstream recordsFile("RecordTable.dat",ios::binary | ios::in | ios::out);

if(!recordsFile)

exit(1);

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

if(4 - i == number) {

recordsFile.seekp((4 - i) \* sizeof(Records));

recordsFile.write(reinterpret\_cast< const char\* >(&player),

sizeof(Records));

recordsFile.close();

return;

}

recordsFile.seekp((3 - i) \* sizeof(Records));

recordsFile.read(reinterpret\_cast< char\* >(&object),

sizeof(Records));

recordsFile.write(reinterpret\_cast< const char\* >(&object),

sizeof(Records));

}

}

//menu.h

#ifndef MENU\_H

#define MENU\_H

#include <iostream>

#include <SFML/Graphics.hpp>

#include "records.h"

#include "camera.h"

using namespace sf;

class Menu {

Sprite menu[4], info, menuBackground, name;

Texture menuTexture[4], infoTexture, menuBackgroundTexture, nameTexture;

bool isMenu;

int numMenu;

void setPosition();

void setColor();

void draw(RenderWindow &window);

void setNumMenu(RenderWindow &window);

public:

Menu(): isMenu(1), numMenu(0) {

menuTexture[0].loadFromFile("images/NG.png");

menuTexture[2].loadFromFile("images/Ab.png");

menuTexture[3].loadFromFile("images/Ex.png");

menuTexture[1].loadFromFile("images/Rec.png");

infoTexture.loadFromFile("images/informationBackground.png");

menuBackgroundTexture.loadFromFile("images/MB4.png");

nameTexture.loadFromFile("images/Name.png");

for(int i = 0; i < 4; i++)

menu[i].setTexture(menuTexture[i]);

info.setTexture(infoTexture), menuBackground.setTexture(menuBackgroundTexture), name.setTexture(nameTexture);

}

int show(RenderWindow &window);

};

#endif

//menu.cpp

#include "menu.h"

void Menu::setPosition() {

float pos = view.getCenter().x - 400;

for(int i = 20, j = 0; i < 150; i += 40, j++)

menu[j].setPosition(pos + 20, i);

info.setPosition(pos, -20);

menuBackground.setPosition(pos + 0, 0);

name.setPosition(pos + 15, 500);

}

void Menu::setColor() {

for(int i = 0; i < 4; i++)

menu[i].setColor(Color(255, 215, 0));

name.setColor(Color(255, 215, 0));

}

void Menu::draw(RenderWindow &window) {

window.draw(menuBackground);

for(int i = 0; i < 4; i++)

window.draw(menu[i]);

window.draw(name);

}

void Menu::setNumMenu(RenderWindow &window) {

if(IntRect(20, 20, 281, 31).contains(Mouse::getPosition(window))) {

menu[0].setColor(Color(255, 140, 0)); numMenu = 1;

}

if(IntRect(20, 100, 184, 27).contains(Mouse::getPosition(window))) {

menu[2].setColor(Color(255, 140, 0)); numMenu = 3;

}

if(IntRect(20, 60, 287, 33).contains(Mouse::getPosition(window))) {

menu[1].setColor(Color(255, 140, 0)); numMenu = 2;

}

if(IntRect(20, 140, 116, 27).contains(Mouse::getPosition(window))) {

menu[3].setColor(Color(255, 140, 0)); numMenu = 4;

}

if(IntRect(15, 500, 770, 29).contains(Mouse::getPosition(window))) {

name.setColor(Color(255, 140, 0));

}

}

int Menu::show(RenderWindow &window) {

setPosition();

while(isMenu) {

//window.clear(Color::Black);

setColor();

numMenu = 0;

setNumMenu(window);

if(Mouse::isButtonPressed(Mouse::Left)) {

switch(numMenu) {

case 1: isMenu = false; break;

case 3: {

window.clear(Color(200, 193, 174));

window.draw(info); window.display();

while(!Keyboard::isKeyPressed(Keyboard::Escape));

break;

}

case 4: window.close(); exit(1);

case 2: {

Records::showRecords(window);

while(!Keyboard::isKeyPressed(Keyboard::Escape));

break;

}

}

}

if(isMenu) {

draw(window); window.display();

}

}

isMenu = true;

}

//camera.h

#ifndef CAMERA\_H

#define CAMERA\_H

#include <SFML/Graphics.hpp>

using sf::View;

extern View view;

extern void setPlayerCoordinateForView(float x, float y);

#endif

//camera.cpp

#include "camera.h"

View view(sf::FloatRect(0, 0, 800, 576));

void setPlayerCoordinateForView(float x, float y) {

float tempX, tempY = 288;

if(x < 400) tempX = 400;

if(x >= 400 && x <= 1520) tempX = x;

if(x > 1520) tempX = 1520;

view.setCenter(tempX, tempY);

}

//cstring.h

#ifndef CSTRING\_H

#define CSTRING\_H

#include <iostream>

#include <iomanip>

#include <conio.h>

#include <Windows.h>

using namespace std;

class cString {

char\* string;

int size;

public:

cString() {

size = 80;string = new char[size];

strcpy(string, "");

}

cString(char \*str) {

size = strlen(str) + 1;string = new char[size];

strcpy(string, str);

}

cString(const char str[]) {

size = strlen(str) + 1;string = new char[size];

strcpy(string, str);

}

cString(const cString& str) {

size = str.size + 1;string = new char[size];

strcpy(string, str.string);

}

cString& operator= (cString& );

cString& operator= (char\* );

cString& operator+= (const cString& );

cString operator+ (const cString& );

cString operator+ (const char\* );

char& operator[] (const int& );

char\* operator() (int , int );

bool operator== (const cString& );

bool operator== (const char\* );

bool operator!= (const cString& );

void operator++ ();

void operator++ (int);

void operator-- (int);

friend cString operator+ (char \*, cString&);

friend ostream& operator<< (ostream&, const cString&);

friend istream& operator>> (istream&, cString&);

int empty( );

int strlen( );

int strlen(const char\* );

const char\* str();

~cString() {

delete [] string;

}

};

#endif

//cstring.cpp

#include "cstring.h"

cString& cString::operator= (cString& obj) {

delete [] this->string;

string = new char[obj.strlen() + 1];

strcpy(this->string, obj.string);

this->size = strlen(this->string);

return \*this;

}

cString& cString::operator= (char\* str) {

if(this->string != nullptr)

this->~cString();

string = new char[size = strlen(str) + 1];

memcpy(this->string, str, size);

return \*this;

}

cString cString::operator+ (const cString& obj) {

cString temp;

temp.string = new char[temp.size = strlen(this->string) +

strlen(obj.string) + 1];

strcpy(temp.string, this->string);

strcat(temp.string, obj.string);

return temp;

}

cString cString::operator+ (const char\* str) {

cString temp;

temp.string = new char[temp.size = strlen(this->string)+strlen(str)+1];

strcpy(temp.string, this->string);

strcat(temp.string, str);

return temp;

}

cString& cString::operator+= (const cString& obj) {

int SIZE = size + obj.size;

char\* str = new char[SIZE];

strcpy(str, this->string);

delete [] this->string;

this->string = str;

strcpy(this->string + size, obj.string);

size = SIZE;

return \*this;

}

cString operator+ (char \*str, cString& obj) {

cString temp;

temp.string = new char[temp.size = strlen(obj.string)+strlen(str)+1];

strcpy(temp.string, str);

strcat(temp.string, obj.string);

return temp;

}

char& cString::operator[] (const int& index) {

if(index <= 0 || index > this->size)

return string[0];

return this->string[index];

}

char\* cString::operator() (int start, int finish) {

if(start < 0 || finish > strlen(this->string))

return "Error in indices";

if(start >= finish) return "Error in indices";

if(this->empty())

return "String is empty";

else {

char \*str;

int size = finish - start;

str = new char[size];

int j = 0;

for(int i = start-1; i < finish; i++, j++)

str[j] = this->string[i];

str[j] = '\0';

return str;

}

}

bool cString::operator== (const cString& obj) {

if(this->size == obj.size) {

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

if(this->string[i] != obj.string[i])

return false;

}

}

return true;

}

bool cString::operator== (const char\* str) {

if(this->size == strlen(str)) {

for(int i = 0; i < this->size; i++) {

if(this->string[i] != str[i])

return false;

}

}

return true;

}

bool cString::operator!= (const cString& obj) {

bool BUFF = false;

if(this->size != obj.size) return true;

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

if(this->string[i] != obj.string[i])

BUFF = true;

}

return BUFF ? true : false;

}

void cString::operator++ () {

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

string[i]++;

}

void cString::operator++ (int) {

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

string[i]++;

}

void cString::operator-- (int) {

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

string[i]--;

}

ostream& operator<< (ostream& os, const cString& obj) {

return os << obj.string;

}

istream& operator>> (istream& is, cString& obj) {

is >> obj.string;

obj.size = obj.strlen();

return is;

}

int cString::empty() {

if(this->size == 0 ) return 1;

else return 0;

}

int cString::strlen() {

int size = 0;

for (; this->string[size] != '\0'; size++);

return size;

}

int cString::strlen(const char\* str) {

int size = 0;

for (; str[size] != 0; size++);

return size;

}

const char\* cString::str(){

char\* buffer = new char(this->size );

memcpy(buffer, this->string, this->size + 1);

return (const char\*)buffer;

}

//main.cpp

#include <iostream>

#include <sstream>

#include <list>

#include <SFML/Graphics.hpp>

#include "cstring.h"

#include "entity.h"

#include "player.h"

#include "enemy.h"

#include "bullet.h"

#include "movingPlatform.h"

#include "menu.h"

#include "map.h"

#include "camera.h"

using namespace sf;

using std::list;

using std::string;

int game(Map& Level, RenderWindow& window);

int main() {

RenderWindow window(VideoMode(800, 576), "Lord of the Rings");

Image mapImage; mapImage.loadFromFile("images/taily\_map.png");

Map Level(mapImage); Level++;

Menu gameMenu; gameMenu.show(window);

while(1) {

if(game(Level, window)) {

Level++;

}

else gameMenu.show(window);

}

return 0;

}

int game(Map& Level, RenderWindow& window) {

Font font; font.loadFromFile("Applications/a\_font\_with\_serifs.ttf");

Text scoreText("", font, 25); scoreText.setColor(Color::Red);

scoreText.setStyle(Text::Bold);

Text buffer("", font, 30); buffer.setColor(Color(255, 215, 0));

buffer.setStyle(Text::Bold);

String str;

Text playerNameText(str, font, 40); playerNameText.setColor(Color::Red);

playerNameText.setStyle(Text::Bold);

Image backgroundImage;

backgroundImage.loadFromFile("images/background0.png");

Texture backgroundTexture;

backgroundTexture.loadFromImage(backgroundImage);

Sprite backgroundSprite; backgroundSprite.setTexture(backgroundTexture);

Image playerImage; playerImage.loadFromFile("images/players11.png");

Player player(70, 240, 32, 49, "Player1", playerImage);

list< Entity\* > entityList;

list< Entity\* >::iterator entityIterator;

Image enemyImage; enemyImage.loadFromFile("images/players.png");

Image movingPlatformImage;

movingPlatformImage.loadFromFile("images/movingPlatform.png");

Image bulletImage; bulletImage.loadFromFile("images/bullet.png");

bulletImage.createMaskFromColor(Color(0, 0, 0));

Level.entitiesCreating(entityList, enemyImage, movingPlatformImage);

Clock clock, gameTimeClock;

float currentFrame = 0, bulletTime = 0;

int gameTime = 0, flag = 0;

while (window.isOpen()) {

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

if(player.getLife()) (int)gameTime =

gameTimeClock.getElapsedTime().asSeconds();

clock.restart();

time = time/800;

bulletTime += time;

Event event;

while (window.pollEvent(event)) {

switch(event.type) {

case Event::Closed:

window.close();

break;

case Event::TextEntered:

if(event.text.unicode >= 32 &&

event.text.unicode <= 126)

str += static\_cast< char >(event.text.unicode);

else if(event.text.unicode == 8 && str.getSize() > 0)

str.erase(str.getSize() - 1, str.getSize());

playerNameText.setString(str);

playerNameText.setPosition(view.getCenter().x +

50, 258);

if(event.text.unicode == 13) {

Records::setResult(str, player.getScore(),

gameTime);

Level.zero();

return 0;

}

break;

}

if (player.isShoot == true) {

player.isShoot = false;

entityList.push\_back(new Bullet

(player.getCoordinateX(),

player.getCoordinateY(), 16, 16,

"PlayerBullet", bulletImage,

player.getState()));

}

}

for (entityIterator = entityList.begin();

entityIterator != entityList.end(); ) {

Entity \*b = \*entityIterator;

b->update(time, currentFrame);

if(!strcmp(b->getName(),"HardEnemy") && bulletTime > 3000) {

entityList.push\_back(new Bullet(

b->getCoordinateX(),

b->getCoordinateY(),

16, 16, "EnemyBullet",

bulletImage, 0));

bulletTime = 0;

}

if (b->getLife() == false) {

entityIterator = entityList.erase(entityIterator);

delete b;

}

else entityIterator++;

}

player.update(time, currentFrame);

player.entitiesCollision(entityList, entityIterator, time);

window.setView(view);

window.clear(Color::White);

ostringstream playerScoreString, playerHealthString,

playerTimeString;

playerScoreString << player.getScore();

playerHealthString << player.getHealth();

playerTimeString << gameTime;

scoreText.setString("Score - " + playerScoreString.str() + " " +

"Health - " + playerHealthString.str() + " " +

"Time - " + playerTimeString.str());

scoreText.setPosition(view.getCenter().x - 350,

view.getCenter().y - 280);

window.draw(backgroundSprite);

Level.draw(window);

window.draw(scoreText);

window.draw(player.getSprite());

for (entityIterator = entityList.begin();

entityIterator != entityList.end(); entityIterator++) {

window.draw((\*entityIterator)->getSprite());

}

if (player.getHealth() <= 0) {

window.clear(Color(200, 193, 174));

buffer.setString("Enter your name: ");

buffer.setPosition(view.getCenter().x - 150, 268);

window.draw(buffer);

window.draw(playerNameText);

}

if(player.getCoordinateX() > 1750 && player.getCoordinateY() < 40)

return 1;

window.display();

}

}