Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: языки программирования(ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«ИГРОВОЕ ПРОГРАММНОЕ СРЕДСТВО SHOOTER»

БГУИР КП 1-40 01 01  012 ПЗ

Студент: гр. 951008 Костюкевич П.Ю.

Руководитель: асс. Марина И.М.

Минск 2020

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Необходимо создать ПС “Shooter”, которое должно реализовывать весь заявленный функционал. ПС должно соответствовать всем требованиям оформления и сопровождения разработки ПО, таким как : четкая постановка задачи , выбор реализуемого функционала, поиск наилучших способов его реализации, создание архитектуры ПС , выбор инструментов для его реализации, создание продукта в соответствии с заданными условиями, написание документации, описывающей созданное ПС.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ…………………………………………………………………………..5

1 АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ И ПОСТАНОВКА ЗАДАЧИ…..6

1.1 Анализ существующих аналогов………………………………………...6

* 1. Постановка задачи………………………………………………………....6

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ………………………………7

2.1 Теоретические сведения по теме…………………………………….…..7

2.2 Математические выкладки……………………………………………….8

2.3 Обоснование выбора языка программирования………………………..9

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА……………….……….9

3.1 Разработка архитектуры ПС……………………………………………..9

3.2 Схемы алгоритмов ПС с описанием……………………………………10

3.3 Разработка схемы работы системы…………………………………….17

3.4 Интерфейсы между компонентами…………………………………….20

3.5 Основные функции разработанного программного обеспечения …...20

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА…………………………..21

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ……………………………………………30

ЗАКЛЮЧЕНИЕ…………………………………………………………………….31

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ………………………………32

ПРИЛОЖЕНИЕ А Код программы……………………………………………….33

**ВВЕДЕНИЕ**

В мире игровой индустрии существует множество жанров игр, таких как : шутеры, RPG, платформеры, симуляторы, стратегии, интеллектуальные игры, гонки и т.д.

В своем курсовом проекте я решил реализовать игру, которая по своей сути является смесью 2-х жанров игр, а именно шутера и платформера. В ней реализованы механики стрельбы ,что сближает ее с шутерами, но в то же время она имеет вид сбоку и большую часть геймплейных механик,присущих платформерам. Данный вид игр до сих пор остается актуальным, это связано с тем ,что данные

игры обычно не требуют большого количества ресурсов компьютера, хотя в то же время играть в них все равно интересно.

Целями курсового проекта являются:

1.Необходимость проведения сравнительного анализа существующих ПС в этой сфере для определения преимуществ и недостатков каждого. Это поможет определить основные требования ,которые должен выполнять разрабатываемый проект.

2. Далее необходимо разработать алгоритмы работы основных структур программы. Это будет описание логики работы окон и их функций ,описание игровой логики, а также увеличенные схемы алгоритмов и связь между ними. Целью данных манипуляций является создание структуры программы.

3. Необходим поиск и анализ основных структур данных ,которые будут использоваться для реализации механик ПС.

4.Необходимо выбрать инструменты для разработки, а также язык программирования.

5.Необходимо реализовать ПС в соответствии со всеми требованиями.

6. После окончания разработки ПС необходимо его протестировать в различных ситуациях и в случае нахождения ошибок, исправить их.

7.При условии корректной работы программы необходимо написать руководство пользователя, которое поможет пользователю научиться работать с ПС и поможет решить различные проблемные ситуации, возникающие при неправильном использовании программы.

**1** **АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ И ПОСТАНОВКА ЗАДАЧИ**

* 1. **Анализ существующих аналогов**

**Игровое ПС “Super Mario Bros.”**

Ссылка на сайт с игрой([www.supermariobros.io](http://www.supermariobros.io)).

Преимущества:

- сюжетный режим;

-анимация объектов;

- хорошо проработанные уровни.

Недостатки:

-старый движок игры;

-пиксельная графика;

-сложность запуска на современных устройствах.

**Игровое ПС “Sonic the Hedgehog”**

Ссылка на сайт с игрой([www. Sonix.tk](http://www.supermariobros.io)).

Преимущества:

- хорошая физика;

-разнообразный геймплей;

- сюжетный режим.

Недостатки:

-cтарый движок игры;

-пиксельная графика;

-cложность запуска на современных устройствах.

* 1. **Постановка задачи**

Подробно изучив такие игровые ПС, как “Super Mario Bros.” И “Sonic the Hedgehog” я пришел к выводу о том , что должна представлять из себя программа, разработанная мною на языке c++

в среде программирования «Microsoft Visual Studio»:

1)Меню игры:

- меню должно содержать кнопку для выхода из игры;

-меню должно отображать текущий счет игрока;

-меню должно содержать кнопку новой игры, для сброса всего прогресса;

-меню должно содержать кнопку выбора уровня для прохождения.

2)Меню выбора уровня:

- меню должно содержать плитки для выбора уровня;

- при прохождении уровней должны открываться новые уровни для прохождения.

3)Уровень и геймплей:

-игрок должен двигаться по карте и “сталкиваться” с препятствиями;

-при убийстве врагов или поднятии монет должны пополняться очки игрока;

-при нажатии на кнопку стрельбы персонаж должен стрелять;

-враги могут стрелять и наносить урон игроку;

-игрок должен взаимодействовать с окружением карты (лестницы, лава шипы и прочее);

-персонаж игрока должен притягиваться к земле;

-должна отображаться шкала здоровья и она должна реагировать на изменение здоровья игрока;

-при пересечении чекпоинта игрок после смерти должен возрождаться на его месте;

-во время нахождения на движущейся платформе игрок должен двигаться вместе с ней.

**2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

**2.1. Теоретические сведения по теме**

Принцип работы данного игрового ПС основан на том, что все графические элементы отображаются в специальном окне. За отображение в нем отвечает графический ускоритель ,который преобразует данные, которые подаются ему в набор пикселей в окне.

Процесс разработки был основан на том, что каждый объект в игре представляет из себя спрайт.

**Спрайт** (англ. **Sprite** — «фея, эльф») — графический объект в компьютерной графике. Чаще всего — растровое изображение, которое можно отобразить на экране.

Каждый спрайт имеет свои размеры, текстуру(изображение), координаты.

Координаты спрайтов рассчитываются относительно общих мировых координат. Они задаются по умолчанию графической библиотекой.

Благодаря тому, что мы имеем представление о размерах и координатах всех спрайтов, мы можем

прописывать взаимодействие всех объектов на карте и отображать их необходимым нам образом.

Для того, чтобы спрайты не были статическими используется основной цикл окна. Его суть заключается в том, что окно обновляет отображаемую информацию с определенной частотой.

Благодаря тому ,что мы можем получить время каждого обновления ,мы можем реализовать обновление игровой логики ,а соответственно и графическое представление в соответствии с реальным временем.

**2.2.Математические выкладки**

Для обработки перемещения и взаимодействия объектов было использовано такое понятие как скорость. В зависимости от скорости и времени обновления окна рассчитываются координаты объектов. Координаты рассчитываются по осям x и y.За координату (0,0) принимается верхняя левая часть окна. Координатные оси направлены вниз(ось Y) и вправо (ось X).

Формулы для расчета:

x = x + dx\*time;

y = y + dy\*time.

Где x, y – координаты объектов, dx, dy –скорость объектов, time – время смены кадра.

Для расчета направления прицела игрока использовались переменные dirX, dirY. Они рассчитываются при помощи косинуса и синуса угла прицеливания соответсвенно . Угол прицеливания изменяется при нажатии определенных клавиш.

Формулы для расчета:

dirSX = cos((aimDeg / 180.0) \* acos(-1)) или dirSX = -cos((aimDeg / 180.0) \* acos(-1)) (в зависимости от направления движения игрока);

dirSY = sin((aimDeg / 180.0) \* acos(-1)).

Где aimDeg – угол прицеливания.

**2.3. Обоснование выбора языка программирования.**

Для разработки игрового ПС был выбран язык программирования с++. Данный язык отлично подходит для разработки игры такого типа ,т.к. он имеет различные библиотеки для работы с графикой .Для разработки данного проекта использовалась библиотека SFML. Она позволяет отрисовывать графические 2d объекты в специальном окне. Данная библитоека отлично подходит для работы , т.к. она остаточно проста в использовании и не занимает большого числа ресурсов компьютера.

Также данный язык является объектно-ориентированным , это означает, что при построении программ можно использовать такие сущности как классы и их объекты. Это особенно удобно для создания игр, т.к. тут происходит работа с сущностями, аналогичными реальным объектам. Помимо этого в данном языке реализована мощная система ссылок и указателей, что позволяет максимально эффективно использовать память компьютера и максимально уменьшить количество лишних операций , что позволяет существенно улучшить производительность программы, что является очень важным параметром при работе с графикой.

Также у данного языка большое сообщество пользователей , что позволяет оперативно находить решения различных проблем .

Для данного языка было разработано множество различных IDE , таких как Visual Studio и Clion.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1. Разработка архитектуры ПС**

1. При запуске программы открывается меню, меню передается окно в котором происходит его отображение. Перед открытием меню происходит загрузка типизированного файла , который хранит различные параметры программы. В меню можно выбрать открытие любого из доступных уровней.

2. При запуске уровня в него передаются параметры загрузки из меню. Каждый уровень является обьектом класса Mission, что облегчает создание большого числа уровней, а также позволяет с легкостью очистить память, отведенную под данный уровень. Ему передается основное окно. Для рисования карты использовалось ПО Tiled , которое позволяет при помощи набора тайлов рисовать карты. Данная программа создает текстовый документ в XML формате. Для загрузки карты мы считываем XML файл при помощи библиотеки TinyXML .Для интерпретации получаемых данных в структуры нашей программы использовался сторонний заголовочный файл Level.h. Далее происходит загрузка карты в проект,используя специальные методы, и ее отрисовка с помощью специального метода. Также из этих загруженных данных мы извлекаем информацию об объектах на карте. Благодаря этой информации мы создаем объекты карты и добавляем их в контейнер для хранения. В качестве контейнера был использован список(list)(см. Рисунок 3.1,3.2), т.к. это достаточно удобная структура для хранения объектов, доступ ко всем объектов которой будет осуществляться одинаково часто и однообразно . Все объекты на карте являются наследники родительского класса Entity. Это было сделано с целью унифицировать обращение ко всем объектам для более легкого взаимодействия с ними.

После выхода из уровня некоторые параметры из него сохраняются в общих параметрах игры.

3.При выходе из игры происходит сохранение всех данных в типизированный файл.

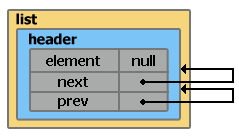


Рисунок 3.1 Пустой список

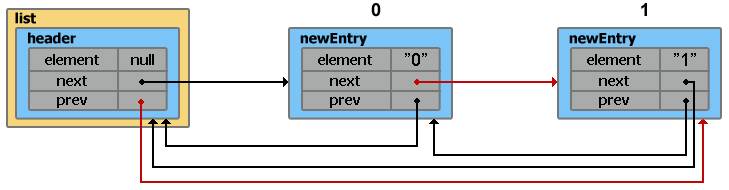


Рисунок 3.2 Пример списка, содержащего элементы

**3.2.Схемы алгоритмов ПС с описанием.**

**3.2.1.Начальное меню**

Открывается окно меню, происходит загрузка данных из файла. Обновляется переменная очков в соответствии с данными из файла. Отображаются кнопки “Новая игра”, “Выбрать уровень”, “Выйти из игры”.

При нажатии на кнопку “Новая игра” происходит сброс параметров игры на значения по-умолчанию.

При нажатии на кнопку “Выбрать уровень” происходит открытие меню выбора уровня.

При нажатии на кнопку “Выйти из игры” происходит выход из игры и закрытие главного окна и сохранение данных игры в файл.



Рисунок 3.3 Укрупненная схема алгоритма главного меню

**3.2.2.Меню выбора уровня**

Открывается окно меню выбора уровня, происходит передача данных для запуска уровней. Отображаются кнопки “Уровень1”, “Уровень2”, “Уровень3”. При наведении мыши на них происходит смена цвета кнопки при условии ,если предыдущий уровень пройден или это первый уровень.

При нажатии на кнопку которая смогла поменять цвет происходит запуск соответствующего уровня и передача в него соответствующих параметров запуска.



Рисунок 3.4 Укрупненная схема алгоритма меню выбора уровня

**3.2.3.Уровень**

При открытии уровня происходит применение соответствующих параметров запуска.

Вначале происходит инициализация всех переменных. Потом происходит загрузка карты. После загрузки карты , все объекты карты загружаются в список состоящий из объектов Entity. После этого программа входит в цикл окна. В нем происходит обновление игровой логики объектов уровня. Также в нем обрабатываются действия пользователя, то есть нажатие клавиш и ответная реакция уровня и персонажа на это. Также постоянно обновляется шкала здоровья персонажа , а также количество набранных очков.

При столкновении с пулей ,врагом , лавой, шипами игрок теряет часть здоровья и временно становиться красным.

Взаимодействие:

-при столкновении с твердыми объектами игрок отталкивается от них;

-при взаимодействии с платформой игрок движется вместе с ней;

-при взаимодействии с лестницей игрок может двигаться вверх и вниз;

-при взаимодействии с монетами игрок получает дополнительные очки;

-при взаимодействии с аптечкой игрок получает определенное количество здоровья;

-при взаимодействии с чекпоинтом персонаж сохраняет свое местоположение в этом месте в случае смерти;

-при падении или прыжке игрок падает вниз(притягивается к земле);

-при движении влево игрок движется влево;

-при движении вправо игрок движется вправо;

-при прицеливании вверх игрок начинает целиться вверх;

-при прицеливании вниз игрок начинает целиться вниз;

-при взаимодействии с твердыми объектами объекты отталкиваются;

-при попадании пули у врага снижается количество здоровья и он временно становиться красным.

Управление:

-при нажатии на клавишу P происходит стрельба;

-при нажатии на клавишу Right происходит движение вправо;

-при нажатии на клавишу Left происходит движение влево;

-при нажатии на клавишу Up происходит прыжок (или поднятие вверх, если игрок на лестнице);

-при нажатии на клавишу Down происходит спуск если игрок на лестнице;

-при нажатии на клавишу W происходит прицеливание вверх;

-при нажатии на клавишу S происходит прицеливание вниз;

-при нажатии клавиши Esc происходит появление окна закрытия уровня;

- при нажатии на кнопку “да” происходит выход из уровня в меню;

- при нажатии на кнопку “нет” происходит закрытие окна закрытия уровня.



Рисунок 3.5 Укрупненная схема алгоритма уровня

**3.3.Разработка схемы работы системы.**

В игровом ПС были реализованы собственные классы разработчика и принадлежащие им методы, а также отдельные методы в заголовочных файлах.

Перечень собственных подпрограмм представлен в таблице 3.1.

Таблица 3.1 Перечень подпрограмм

|  |  |  |  |
| --- | --- | --- | --- |
| Имя метода | Описание | Имя параметра | Назначение параметра |
| Заголовочный файл Menu.h | | | |
| initParams() | Производит загрузку данных из файла и инициализирует переменную params | - | - |
| menu  (RenderWindow& window) | Производит запуск логики меню , а также открытие этого окна | window | Ссылка на объекта окна |
| Заголовочный файл Play.h | | | |
| play  (LoadParams& loadParams, RenderWindow& window) | Производит запуск логики окна выбора уровней, а также производит его открытие, передает параметры для запуска уровней | loadParams  window | Ссылка на параметры для запуска уровней  Ссылка на объект окна |
| Заголовочный файл InitEntities.h | | | |
| initEntities  (Level\* lvl, std::list<Entity\*>& entities) | Производит создание всех объектов уровня и их запись в список объектов уровня | lvl  entities | Указать на объект уровня  Ссылка на список объектов уровня |
| Заголовочный файл UpdateEntities.h | | | |
| updateEntities  (std::list<Entity\*>& entities, float time, Player& p) | Производит обновление параметров всех объектов на карте | entities  p | Ссылка на список объектов уровня  Ссылка на объект игрока |
| Заголовочный файл View.h | | | |
| setPlayerCoordinateForView  (float x, float y,View& view) | Устанавливает вид окна на координаты игрока | x  y  view | Координата x игрока  Координата y игрока  Ссылка на вид окна |
| Заголовочный файл ObjectInteraction.h | | | |
| interact  (float time, std::list<Entity\*>& entities) | Позволяет обработать взаимодействие между всеми объектами карты. | time  entities | Время между сменой кадров  Ссылка на список объектов уровня |
| Заголовочный файл SaveData.h | | | |
| saveData  (LoadParams& params) | Позволяет сохранять данные уровней и игры в файл | params | Ссылка на данные уровней и  игры |
| Заголовочный файл View.h | | | |
| setPlayerCoordinateForView  (float x, float y, View& view) | Позволяет задать координаты отображения для вида. | x  y  view | Координата x  Координата y  Ссылка на вид |
| Класс Entity | | | |
| update(float time) | Позволяет обновлять параметры объектов уровня | time | Время между сменой кадров |
| checkCollisionWithMap  (float dx, float dy) | Позволяет проверять столкновение объектов с твердыми поверхностями уровня | dx  dy | Скорость движения объекта по оси x  Скорость движения объекта по оси y |
| Interaction  (Entity\* obj, float time) | Позволяет осуществлять взаимодействие между объектами уровня | obj  time | Указатель на объект с которым будет обрабатываться взаимодействие  Время между сменой кадров |
| Класс Player наследует класс Entity | | | |
| control() | Позволяет осуществлять обработку нажатий пользователя на клавиатуру и изменение состояния игрока | - | - |
| Fire  (std::list<Entity\*>& entities) | Позволяет осуществлять стрельбу персонажа при нажатии на соответствующую кнопку | entities | Ссылка на список объектов уровня |
| Класс Enemy наследует класс Entity | | | |
| Fire  (list<Entity\*>& entities) | Позволяет осуществлять стрельбу врага | entities | Ссылка на список объектов уровня |
|  |  |  |  |
| Класс Bullet наследует класс Entity | | | |
| Класс Checkpoint наследует класс Entity | | | |
| Класс Boost наследует класс Entity | | | |
| Класс MovingPlatform наследует класс Entity | | | |
| Класс LifeBar | | | |
| update(int health) | Позволяет обновлять шкалу здоровья персонажа | health | Здоровье персонажа |
| Draw  (RenderWindow& window) | Позволяет отрисовывать шкалу здоровья персонажа | window | Ссылка на окно для отображения |

**3.4.Интерфейсы между компонентами.**

При открытии программы происходит запуск функции menu и открывается окно главного меню

При нажатии на кнопку ”Выбрать уровень” происходит запуск функции play и открывается

окно выбора уровня.

После выбора уровня создается объект класса Mission и происходит вызов его функции start, после этого открывается окно с уровнем.

При закрытии окна с уровнем происходит окончание функции start и продолжается выполнение функции menu.

При закрытии окна меню происходит выход из функции menu.

**3.5.Основные функции разработанного программного обеспечения.**

Данное игровое ПС поддерживает различный функционал:

-система уровней;

-сохранение данных прогресса;

-подсчет количества собранных очков;

-система врагов;

-система взаимодействия персонажа с окружающей средой;

-физика персонажа;

-стрельба персонажа;

-перемещение персонажа;

-прицеливание персонажа;

-система чекпоинтов;

-интеграция с ПО Tiled, что позволяет рисовать уровни в специальном ПО.

**4ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Для того, чтобы соответствовать требованиям к проектируемому программному средству, необходимо, чтобы оно прошло некоторое тестирование, способное выявить его недостатки.

Таблица 4.1 Тест-кейсы

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Результат |
| **Отображение главного меню**  1.Запустить программу  2.Проверить элементы окна | 1.Открывается окно меню  2.Отображаются следующие элементы:  - кнопка “Новая игра”;  - кнопка “Выбрать уровень”;  - кнопка “Выйти из игры”;  - текст “Score” с количеством очков рядом. | Успех  См. рисунок 4.1 |
| **Работоспособность кнопки “Новая игра”**  1.Запустить программу  2.Нажать на кнопку “Новая игра” | 1.Количество очков обнуляется  2.Становится доступным только первый уровень | Успех  См. рисунок 4.2 |
| **Работоспособность кнопки “Выйти из игры”**  1.Запустить программу  2.Нажать на кнопку “Выйти из игры” | 1.Происходит закрытие окна с игрой | Успех |
| **Работа сохранения в файл**  1.Запустить программу  2.Нажать на кнопку “Выйти из игры”  3.Проверить файл с именем “params” | 1.В файле в поле Score  Отобразиться такое же количество очков как и в игре.  2.В файле в полях Level  обновятся доступные уровни | Успех  См. рисунок 4.3  См. рисунок 4.4 |
| **Работа загрузки из файла**  1.Запустить программу  2.Проверить файл с именем “params” | 1.В игре отобразиться такое же количество очков как и в файле в поле Score  2.В игре будут доступны такие же уровни как и в файле в полях Level | Успех  См. рисунок 4.3  См. рисунок 4.4 |
| **Работоспособность кнопки “Выбрать уровень”**  1.Запустить программу  2.Нажать на кнопку “Выбрать уровень” | 1.Открывается окно выбора уровня  2.Отображаются следующие элементы:  - кнопка “Уровень 1”;  - кнопка “Уровень 2”;  - кнопка “Уровень 3”. | Успех  См. рисунок 4.5 |
| **Работоспособность кнопок выбора уровня**  1.Запустить окно выбора уровня  2.Выбрать уровень | 1.Открывается окно с уровнем  2.Отображаются следующие элементы:  -элементы уровня;  -количество очков;  -шкала здоровья. | Успех  См. рисунок 4.6 |
| **Работоспособность клавиш управления персонажем**  1.Запустить уровень  2.Нажать клавишу Left  3.Нажать клавишу Right  4.Нажать клавишу Up  5.Нажать клавишу Down  6.Нажать клавишу P  7.Нажать клавишу W  8.Нажать клавишу S | 1.При нажатии клавиши из п.2 происходит перемещение персонажа влево  1.При нажатии клавиши из п.3 происходит перемещение персонажа вправо  1.При нажатии клавиши из п.4 происходит прыжок персонажа(если персонаж находится на лестнице, то он начинает перемещаться вверх по лестнице)  1.При нажатии клавиши из п.5 , если персонаж персонаж находится на лестнице , то он начинает перемещаться вниз по лестнице  1.При нажатии клавиши из п.6 персонаж начинает стрелять патронами  1.При нажатии клавиши из п.7 прицел персонажа начинает перемещаться вверх  1.При нажатии клавиши из п.8 прицел персонажа начинает перемещаться вниз | Успех |
| **Притяжение персонажа к земле при падении**  1.Запустить уровень  2.Совершить падение | 1.Персонаж притягивается к земле | Успех |
| **Столкновение объектов с твердыми поверхностями**  1.Запустить уровень | 1.Персонаж и объекты не должны проваливаться через текстуры | Успех |
| **Столкновение персонажа с шипами, лавой,врагами, пулями**  1.Запустить уровень  2.Произвести все стокновения | 1.Персонаж становиться  красным  2.Шкала здоровья персонажа уменьшается | Успех |
| **Сбор монет**  1.Запустить уровень  2.Взять монету | 1.Монета пропадает  2.Увеличивается количество очков | Успех |
| **Сбор медикаментов**  1.Запустить уровень  2.Взять медикаменты | 1.Медикаменты пропадают  2.Увеличивается шкала здоровья | Успех |
| **Попадание пули в врага**  1.Запустить уровень  2.Пустить пулю во врага | 1.Враг становиться красным  2.Уменьшается здоровье врага | Успех |
| **Смерть врага**  1.Запустить уровень  2.Запустить достаточное количество пуль | 1.Враг исчезает  2.Увеличивается количество очков | Успех |
| **Перемещение персонажа на движущейся платформе**  1.Запустить уровень  2.Встать на платформу | 1.Персонаж перемещается вместе с платформой | Успех |
| **Смерть персонажа**  1.Запустить уровень  2.Получить такой урон, чтобы шкала здоровья стала пустой | 1.Персонаж исчезнет и появится в последней точке сохранения с полной шкалой здоровья | Успех |
| **Пересечение чекпоинта**  1.Запустить уровень  2.Пересечь чекпоинт | 1.Произойдет сохранение в данной точке | Успех |
| **Пересечение точки окончания уровня**  1.Запустить уровень  2.Пересечь точку окончания уровня | 1.Произойдет выход из окна уровня  2.Откроется окно меню | Успех  См. рисунок 4.7  См. рисунок 4.8 |
| **Нажатие клавиши “Escape”**  1.Запустить уровень  2Нажать клавишу | 1.Произойдет открытие меню выхода из уровня | Успех  См. рисунок 4.9 |
| **Нажатие кнопки “**✔**”**  1.Запустить уровень  2Нажать клавишу “Escape”  3.Нажать кнопку “✔ ” | 1.Произойдет выход из уровня  2.Откроется окно меню | Успех  См. рисунок 4.9  См. рисунок 4.10 |
| **Нажатие кнопки “**✖**”**  1.Запустить уровень  2Нажать клавишу “Escape”  3.Нажать кнопку “✖ ” | 1.Произойдет закрытие меню выхода из уровня  2.Уровень продолжит работу | Успех  См. рисунок 4.11  См. рисунок 4.9 |

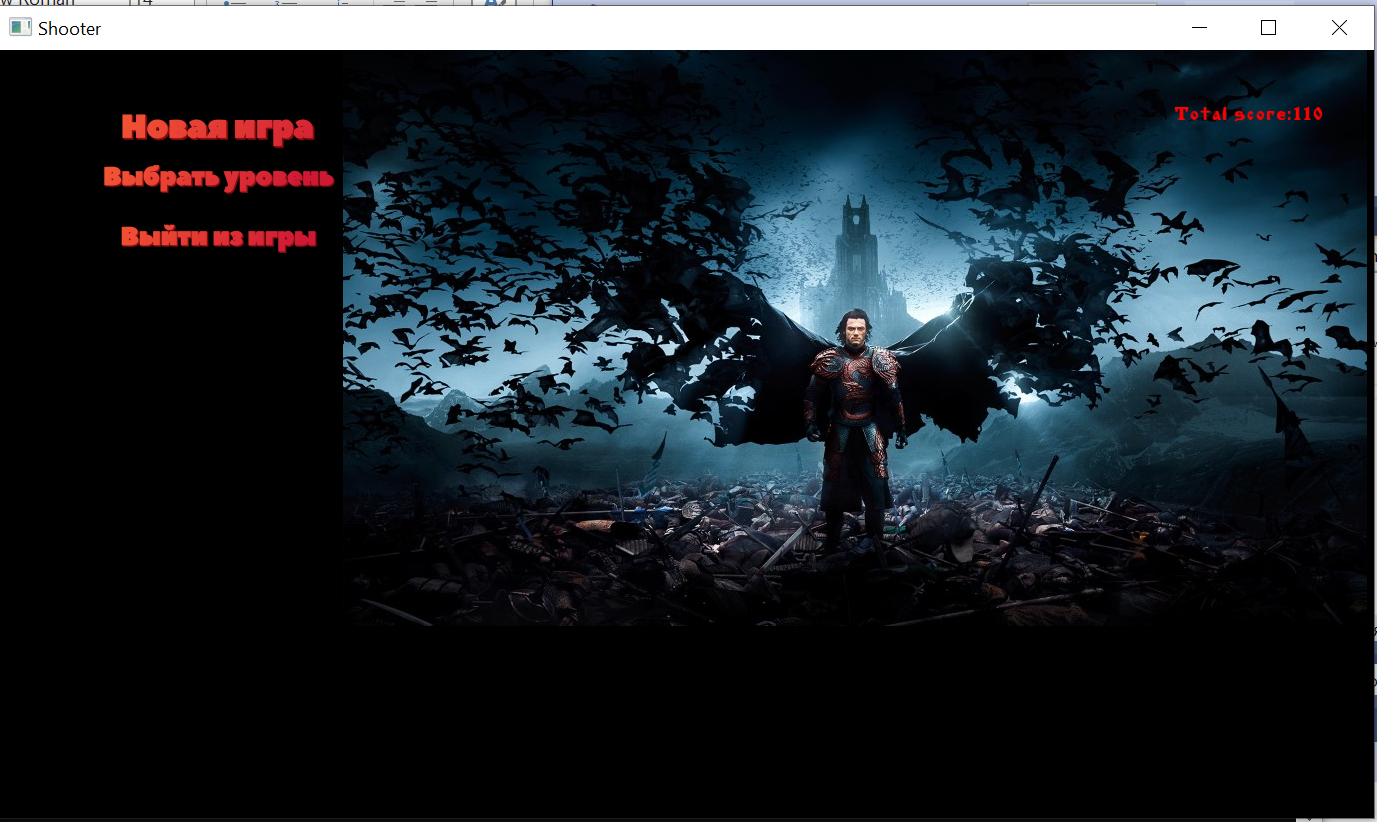


Рисунок 4.1 Отображение главного окна

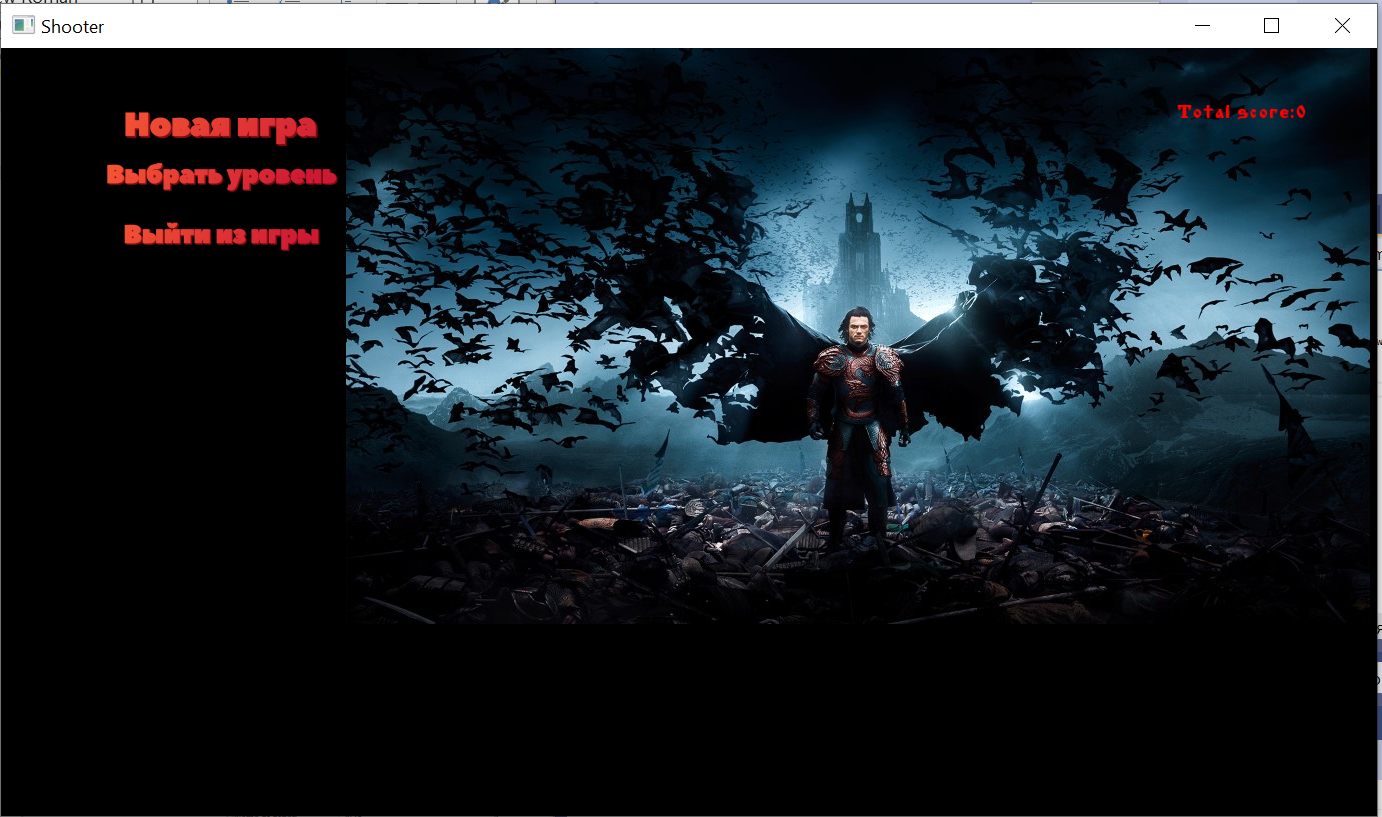
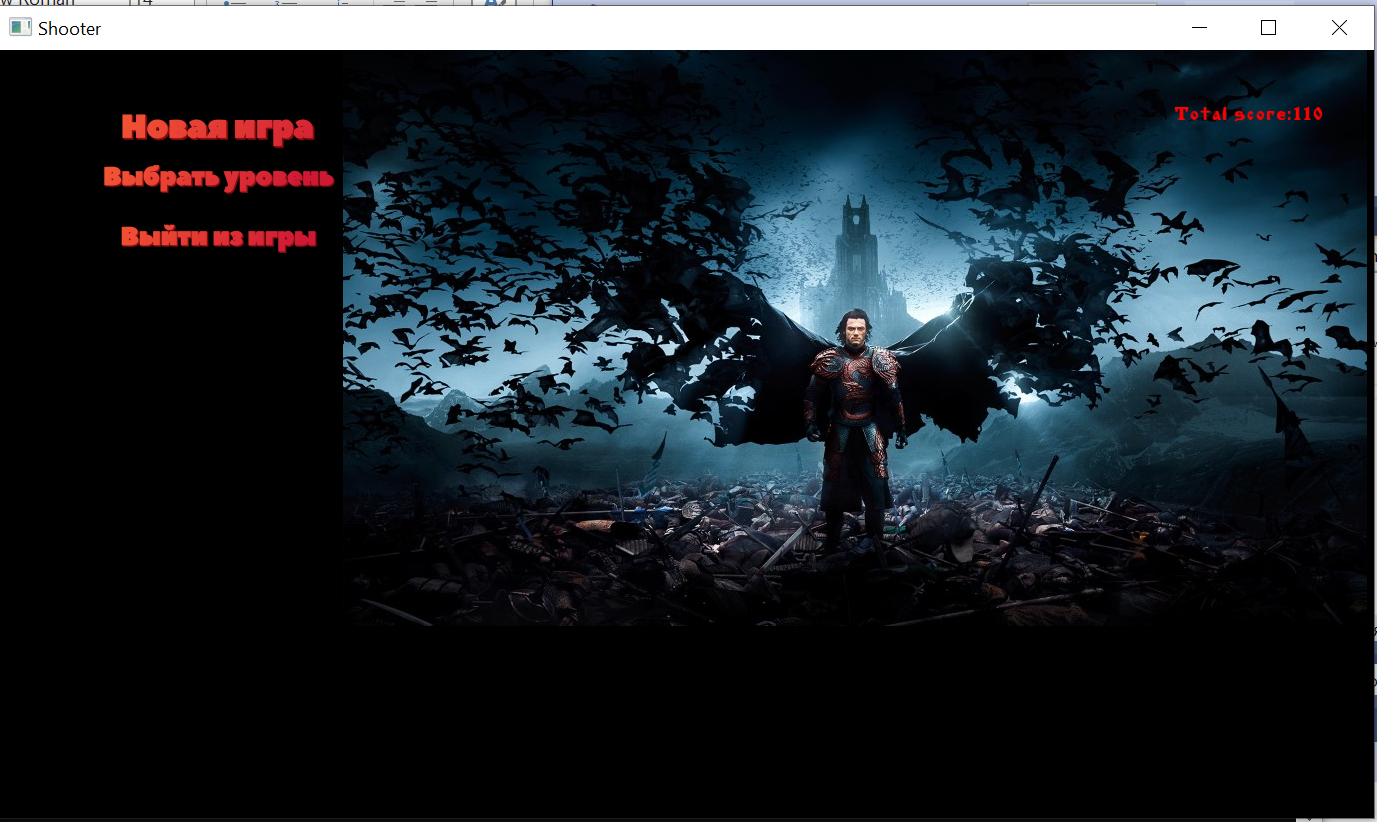
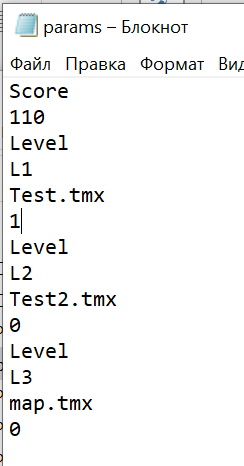
****

Рисунок 4.2 Работоспособность кнопки «Новая игра»



****

Рисунки 4.3 и 4.4 Работа сохранения в файл и работа загрузки из файла

****

Рисунок 4.5 Работоспособность кнопки «Выбрать уровень»

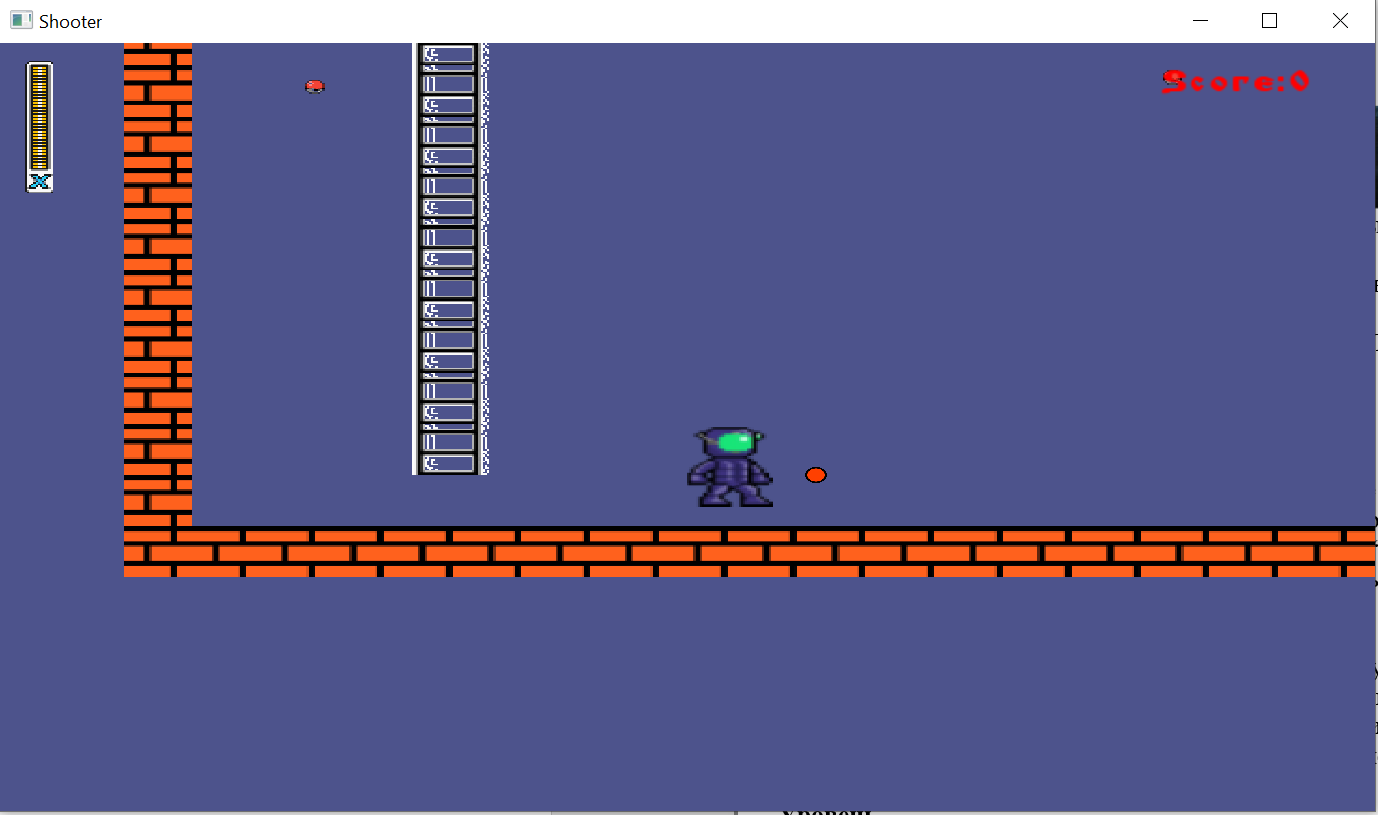
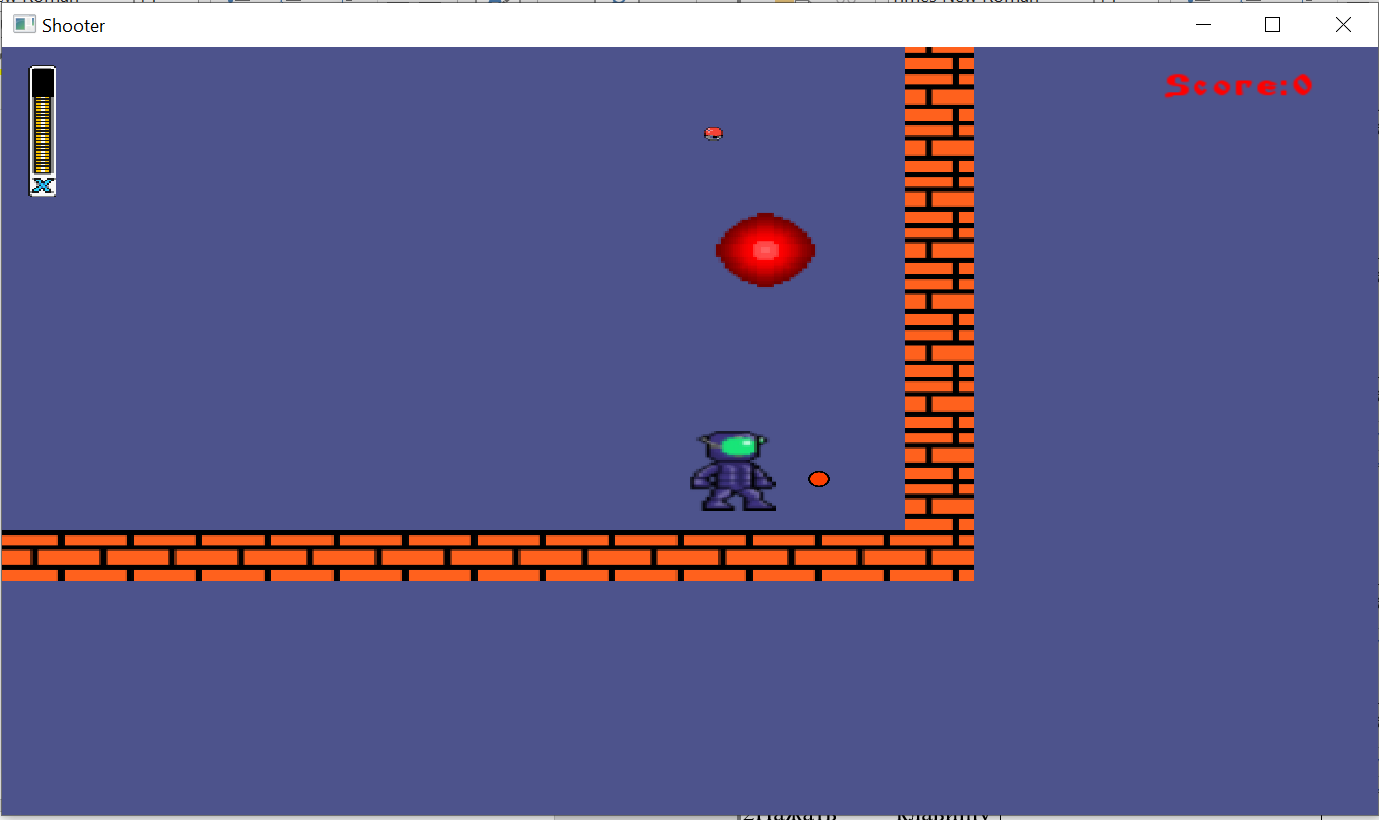
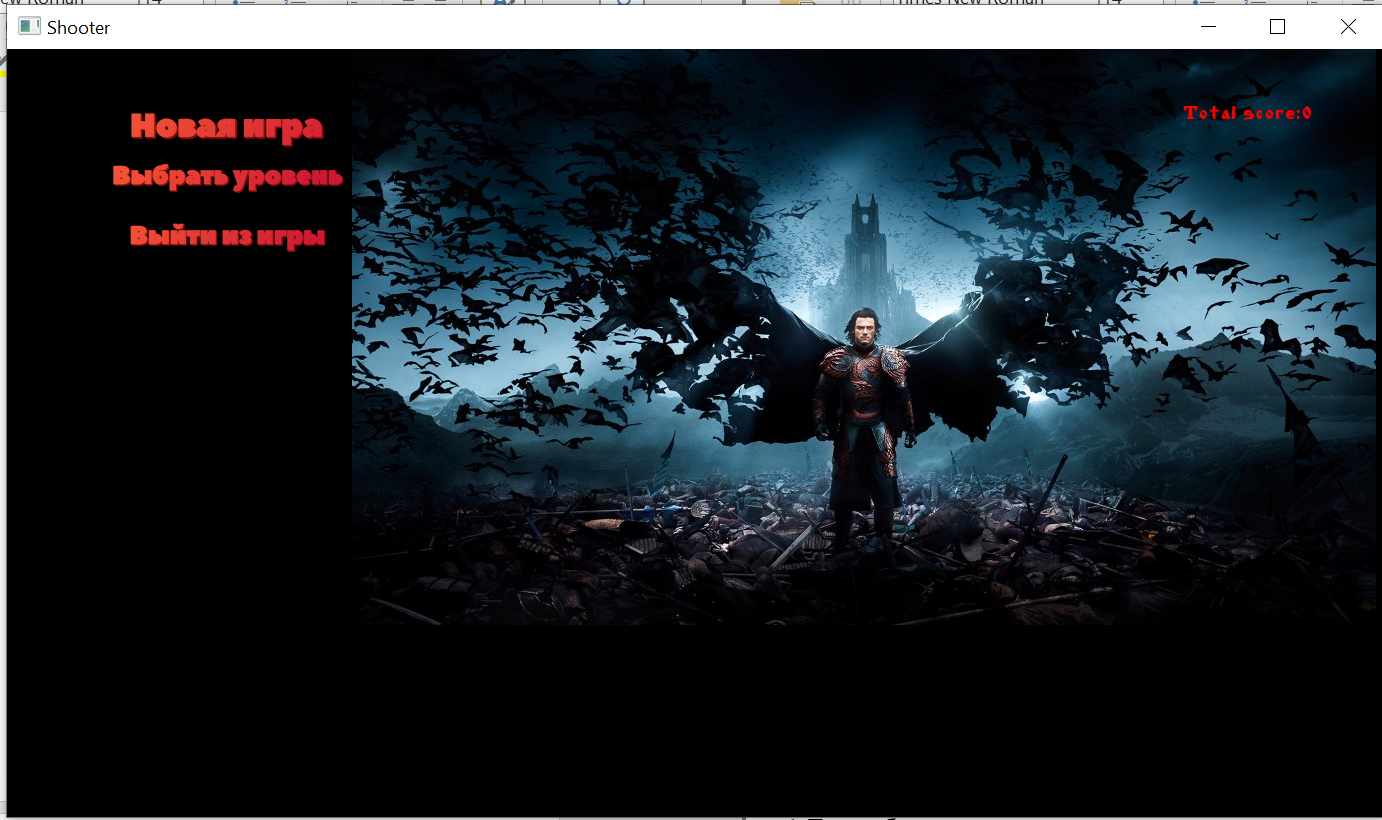
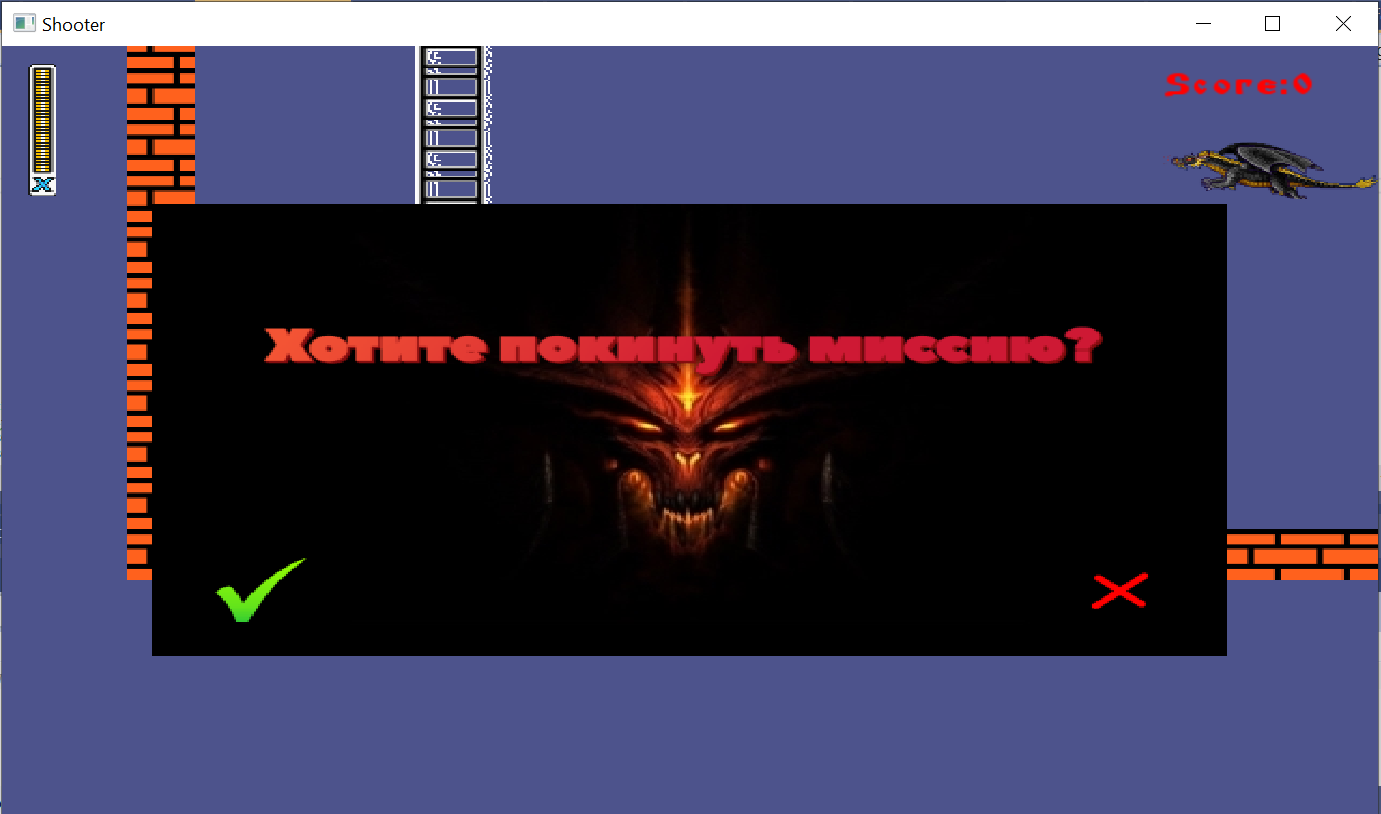
****

Рисунок 4.6 Работоспособность кнопок выбора уровня

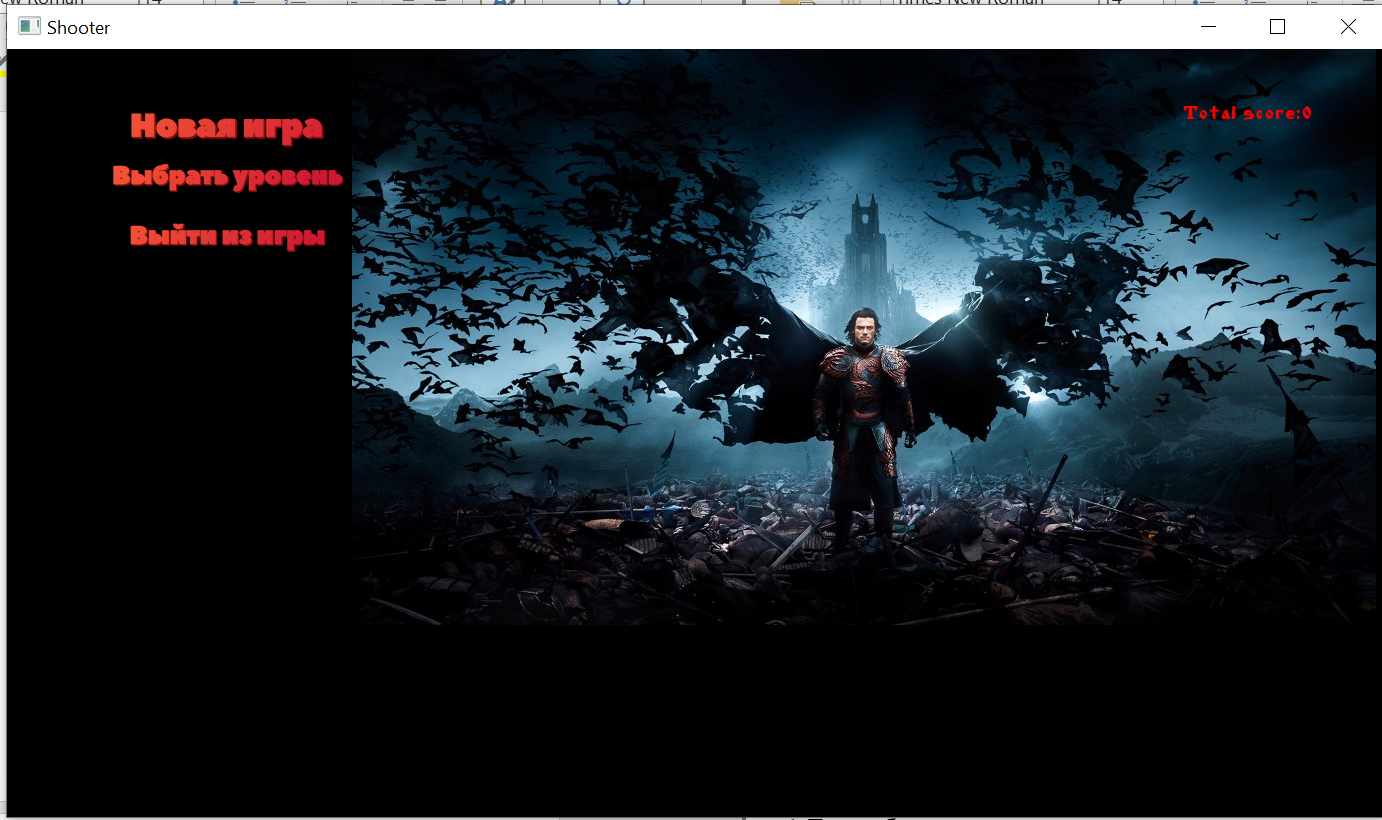
****

****

**Рисунки 4.7, 4.8 Пересечение точки окончания уровня**

****

**Рисунок 4.9 Нажатие клавиши “Escape”**

****

**Рисунок 4.10 Нажатие кнопки “**✔**”**

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

1.Запустить программное средство

**Окно главного меню**

1.Для просмотра количества ваших очков посмотрите на поле Score.

2.Для сброса прогресса нажмите на кнопку “Новая игра”.

3.Для открытия окна выбора уровня нажмите кнопку “Выбрать уровень”.

4.Для выхода из игры нажмите кнопку “Выйти из игры”.

**Окно выбора уровня**

1.Для выбора уровня наведите курсор мыши на кнопку необходимого уровня. Если кнопка не изменила свой цвет, то данный уровень заблокирован. Для разблокировки уровня необходимо пройти предыдущий уровень.

Если кнопка поменяла цвет, то нажмите левой клавишей мыши на эту кнопку.

**Уровень**

Для управления персонажем необходимо использовать следующие клавиши:

-клавиша “Left”- для перемещения влево;

-клавиша “Right”- для перемещения вправо;

-клавиша “Down”-для перемещения вниз по лестнице;

-клавиша “Up”-для прыжка или подъема по лестнице;

-клавиша “P”-для стрельбы;

-клавиша “W”-для прицеливания вверх;

-клавиша “S”-для прицеливания вниз.

Во время прохождения будут происходить сохранения при пересечении чекпоинтов.

При столкновении с лавой, врагами, шипами, пулями будет происходить понижение шкалы здоровья. Когда шкала здоровья закончится ваш персонаж умрет и возродится в точке сохранения.

Для убийства врагов стреляйте по ним пока они не умрут.

Для набора очков убивайте врагов и собирайте монеты.

Для пополнения здоровья собирайте медикаменты.

Для прохождения уровня необходимо достичь конечной точки.

Для выхода из уровня необходимо нажать клавишу “Escape” и нажать кнопку “✔ ”.Для отмены действия нажмите кнопку “✖ ”.

**Хорошего времяпрепровождения!!!**

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта было создано исправно работающее игровое ПС “Shooter”, которое может найти применение в различных местах, в основном в сфере развлечений.

В ходе разработки было исследовано несколько похожих программных средств. В итоге были сформулированы основные требования для программы такого типа:

-система уровней;

-система меню;

-система прогресса и его сохранение;

-система построения карты;

-система управления персонажем;

-система логики всех объектов на карте.

Далее были разработаны алгоритмы работы различных окон, реализующих функционал ПС.

Разработка велась на языке с++ с использованием библиотеки SFML для работы с графикой. В качестве основы данного способа разработки выступили окна, способные отображать спрайты. Каждый спрайт имеет свой набор параметров, который позволяет реализовывать логику взаимодействия всех спрайтов друг с другом. Также для обновления этой логики использовался основой цикл окна в котором происходило отображение всех спрайтов.

В качестве основной структуры данных был выбран двунаправленный связанный список (List).

В ходе процесса создания игрового ПС основные функции были детализированы.

Далее проводилось тестирование работы программы в различных ситуациях.

В конце было разработано руководство пользователя, для обучения пользователей правильной работе с данной программой.

**Вывод:**

в результате работы было создано игровое ПС, которое имеет достаточный набор функционала, чтобы соответствовать ПС такого класса.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] kychka-pc.ru [Электронный портал] – Электронные данные. – Режим доступа: <https://kychka-pc.ru/>

[2] metanit.com [Электронный портал] – Электронные данные. – Режим доступа: <https://metanit.com/cpp/tutorial/>

**ПРИЛОЖЕНИЕ А**

**Код программы**

**Boost.cpp**

#include "Boost.h"

Boost::Boost(Image& image, String name, Level& lvl, float x, float y, int w, int h) :Entity(image, name, x, y, h, w, 0)

{//всё так же, только взяли в конце состояние игрока (int dir)

obj = lvl.GetObjects("solid");

life = true;

if (name == "Money")

{

score = 20;

}

}

void Boost::interaction(Entity\* obj, float time)

{

if ((obj->name == "Player") && (obj->getRect().intersects(this->getRect())))

{

life = false;

}

}

void Boost::checkCollisionWithMap(float dx, float dy)

{

}

void Boost::update(float time)

{

sprite.setPosition(x + w / 2, y + h / 2);

}

**Boost.h**

#pragma once

#include "Entity.h"

class Boost :

public Entity

{

public:

Boost(Image& image, String name, Level& lvl, float x, float y, int w, int h);

void interaction(Entity\* obj, float time);

void checkCollisionWithMap(float dx, float dy);

void update(float time);

};

**Bullet.cpp**

#include "Bullet.h"

#include <iostream>

Bullet::Bullet(Image& image, String name, Level& lvl, float x, float y, int w, int h, float speed, float dirX, float dirY) :Entity(image, name, x, y, h, w, 0) {//всё так же, только взяли в конце состояние игрока (int dir)

obj = lvl.GetObjects("solid");//инициализируем .получаем нужные объекты для взаимодействия пули с картой

this->dirX = dirX;

score = 0;

this->dirY = dirY;

this->speed = speed;

life = true;

float lenght = sqrt(dirY \* dirY + dirX \* dirX);

dx = (dirX / lenght) \* speed;

dy = (dirY / lenght) \* speed;

//выше инициализация в конструкторе

}

void Bullet::update(float time)

{

x += dx \* time;//само движение пули по х

y += dy \* time;//по у

for (int i = 0; i < obj.size(); i++) {//проход по объектам solid

if (getRect().intersects(obj[i].rect)) //если этот объект столкнулся с пулей,

{

life = false;// то пуля умирает

}

}

sprite.setPosition(x + w / 2, y + h / 2);//задается позицию пуле

}

void Bullet::interaction(Entity\* obj, float time)

{

if (((obj->name == "Player" && name == "EBullet") || obj->name == "MovingPlatform" || ((obj->name == "Enemy2" || obj->name == "Enemy3" || obj->name == "EasyEnemy") && name == "Bullet")) && (obj->getRect().intersects(this->getRect()))) //если этот объект столкнулся с пулей,

{

life = false;// то пуля умирает

}

}

void Bullet::checkCollisionWithMap(float dx, float dy)

{

}

**Bullet.h**

#pragma once

#include "Entity.h"

class Bullet :

public Entity

{

public:

float dirX;//направление пули

float dirY;

int score;

Bullet(Image& image, String name, Level& lvl, float x, float y, int w, int h, float speed, float dirX, float dirY);

void update(float time);

void interaction(Entity\* obj, float time);

void checkCollisionWithMap(float dx, float dy);

};

**Checkpoint.cpp**

#include "Checkpoint.h"

void Checkpoint::interaction(Entity\* obj, float time)

{

if ((obj->name == "Player") && (obj->getRect().intersects(this->getRect())))

{

}

}

void Checkpoint::update(float time)

{

sprite.setPosition(x + w / 2, y + h / 2);

}

void Checkpoint::checkCollisionWithMap(float dx, float dy)

{

}

Checkpoint::Checkpoint(Image& image, String name, Level& lvl, float x, float y, int w, int h) :Entity(image, name, x, y, h, w, 0)

{

life = true;

}

**Checkpoint.h**

#pragma once

#include "Entity.h"

class Checkpoint :

public Entity

{

public:

Checkpoint(Image& image, String name, Level& lvl, float x, float y, int w, int h);

void interaction(Entity\* obj, float time);

void update(float time);

void checkCollisionWithMap(float dx, float dy);

};

**Enemy.cpp**

#include "Enemy.h"

#include <list>

#include "Bullet.h"

Enemy::Enemy(Image& image, String name, Level& lvl, float x, float y, int w, int h, list<Entity\*>& entities) :Entity(image, name, x, y, h, w, 0) {

obj = lvl.GetObjects("solid");//инициализируем.получаем нужные объекты для взаимодействия врага с картой

level = lvl;

this->entities = &entities;

if (name == "EasyEnemy") {

sprite.setTextureRect(IntRect(0, 0, w, h));

speed = 0.1;

dx = speed;

//dy = speed;

score = 20;

}

if (name == "Enemy2")

{

speed = 0.1;

lx = 0;

ly = 0;

dx = speed;

dy = 0;

score = 40;

}

if (name == "Enemy3")

{

lx = 0;

ly = 0;

speed = 0.1;

dx = 0;

dy = speed;

score = 30;

}

damageTimer = 100;

isDamaged = false;

}

void Enemy::checkCollisionWithMap(float dx, float dy)

{

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

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

{

if (obj[i].name == "solid")//если встретили препятствие

{

if (dy > 0) {

if (name == "EasyEnemy")

{

y = obj[i].rect.top - h; this->dy = 0; onGround = true;

}

if (name == "Enemy2")

{

y = obj[i].rect.top - h; this->dy = -speed;

ly = 0;

}

if (name == "Enemy3")

{

y = obj[i].rect.top - h; this->dy = -speed;

ly = 0;

}

}

if (dy < 0) {

if (name == "EasyEnemy") {

y = obj[i].rect.top + obj[i].rect.height; this->dy = 0;

}

if (name == "Enemy2")

{

y = obj[i].rect.top + obj[i].rect.height; this->dy = speed;

ly = 0;

}

if (name == "Enemy3")

{

y = obj[i].rect.top + obj[i].rect.height; this->dy = speed;

ly = 0;

}

}

if (dx > 0) {

if (name == "EasyEnemy") {

x = obj[i].rect.left - w; this->dx = -0.1; sprite.scale(-1, 1);

}

if (name == "Enemy2")

{

x = obj[i].rect.left - w; this->dx = -speed;

lx = 0;

sprite.setTextureRect(IntRect(w, 0, -w, h));

}

}

if (dx < 0)

{

if (name == "EasyEnemy") {

x = obj[i].rect.left + obj[i].rect.width; this->dx = 0.1; sprite.scale(-1, 1);

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "Enemy2")

{

x = obj[i].rect.left + obj[i].rect.width; this->dx = speed;

lx = 0;

}

}

}

}

}

void Enemy::update(float time)

{

checkCollisionWithMap(dx, dy);

if (name == "EasyEnemy") {//меняет направление примерно каждые 3 сек(альтернативная версия смены направления)

x += dx \* time;

}

if (name == "Enemy2") {

if (lx <= 100 && dx > 0)

{

dx = speed;

dy = 0;

x += dx \* time;

lx += dx \* time;

}

else if (dx > 0)

{

dy = speed;

ly = 0;

dx = 0;

}

else if (ly <= 100 && dy > 0)

{

dx = 0;

y += dy \* time;

ly += dy \* time;

}

else if (dy > 0)

{

sprite.setTextureRect(IntRect(w, 0, -w, h));

dx = -speed;

lx = 0;

dy = 0;

}

else if (lx >= -100 && dx < 0)

{

dy = 0;

x += dx \* time;

lx += dx \* time;

}

else if (dx < 0)

{

dy = -speed;

ly = 0;

dx = 0;

}

else

if (ly >= -100 && dy < 0)

{

dx = 0;

y += dy \* time;

ly += dy \* time;

}

else if (dy < 0)

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = speed;

lx = 0;

dy = 0;

}

sprite.setPosition(x + w / 2, y + h / 2);

}

if (name == "Enemy3") {

checkCollisionWithMap(dx, dy);

if (ly <= 100 && dy > 0)

{

y += dy \* time;

ly += dy \* time;

}

else if (dy > 0)

{

dy = -speed;

}

else if (ly >= -100 && dy < 0)

{

y += dy \* time;

ly += dy \* time;

}

else if (dy < 0)

{

dy = speed;

}

}

fireTimer += time;

if (fireTimer > 1000) {

fireTimer = 0;

fire(\*entities);

}

if (isDamaged)

{

if (damageTimer > 0)

{

sprite.setColor(Color::Red);

damageTimer -= time;

}

else

{

sprite.setColor(Color::White);

isDamaged = false;

damageTimer = 100;

}

}

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

sprite.setPosition(x + w / 2, y + h / 2);

}

void Enemy::interaction(Entity\* obj, float time)

{

if ((this->getRect().intersects(obj->getRect())) && (this->name == "EasyEnemy") && (obj->name == "EasyEnemy"))//если столкнулись два объекта и они враги

{

this->dx \*= -1;//меняем направление движения врага

this->sprite.scale(-1, 1);//отражаем спрайт по горизонтали

}

if ((this->getRect().intersects(obj->getRect())) && (obj->name == "Bullet"))

{

if (!isDamaged) {

isDamaged = true;

}

health -= 20;

if (health < 0)

{

obj->score = this->score;

}

}

}

void Enemy::fire(list<Entity\*>& entities)

{

Image BulletImage;//изображение для пули

BulletImage.loadFromFile("images/bullet.png");//загрузили картинку в объект изображения

BulletImage.createMaskFromColor(Color::White);//маска для пули по черному цвету

if (name == "EasyEnemy")

{

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x + 30, y, 16, 16, 0.8, x, 0));

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x - 30, y, 16, 16, 0.8, -x, 0));

}

if (name == "Enemy2")

{

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x, y, 16, 16, 0.8, x, 0));

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x, y, 16, 16, 0.8, -x, 0));

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x, y, 16, 16, 0.8, 0, y));

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x, y, 16, 16, 0.8, 0, -y));

}

if (name == "Enemy3")

{

float dirSX, dirSY;

for (int aimDeg = 0; aimDeg < 360; aimDeg += 36)

{

dirSX = cos((aimDeg / 180.0) \* acos(-1));

dirSY = sin((aimDeg / 180.0) \* acos(-1));

(entities).push\_back(new Bullet(BulletImage, "EBullet", level, x, y, 16, 16, 0.8, dirSX, dirSY));

}

}

}

**Enemy.h**

#pragma once

#include "Entity.h"

#include <list>

class Enemy :

public Entity

{

public:

list<Entity\*>\* entities;

Level level;

float ly;

bool isDamaged;

int damageTimer;

float lx;

Enemy(Image& image, String name, Level& lvl, float x, float y, int w, int h, list<Entity\*>& entities);

void checkCollisionWithMap(float dx, float dy);

void update(float time);

void interaction(Entity\* obj, float time);

void fire(list<Entity\*>& entities);

};

**Entity.cpp**

#include "Entity.h"

Entity::Entity(Image& image, String name, float x, float y, int h, int w, int score) {

this->name = name; moveTimer = fireTimer = 0; this->x = x; this->y = y; this->h = h; this->w = w; this->score = score;

speed = 0; health = 100; dx = 0; dy = 0;

life = true; onGround = false; isMove = false;

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setOrigin(w / 2, h / 2);

}

FloatRect Entity::getRect() {

return FloatRect(x, y, w, h);

}

void Entity::update(float time)

{

}

void Entity::checkCollisionWithMap(float dx, float dy)

{

}

void Entity::interaction(Entity\* obj, float time)

{

}

**Entity.h**

#pragma once

#include <SFML/Graphics.hpp>

#include <vector>

#include "level.h"

#include <iostream>

using namespace std;

using namespace sf;

class Entity

{

public:

std::vector<Object> obj;

float dx, dy, x, y, speed, moveTimer, fireTimer;

int w, h, health, score;

bool life, isMove, onGround;

Texture texture;

float currentFrame;

Sprite sprite;

string name;

Entity(Image& image, String name, float x, float y, int h, int w, int score);

FloatRect getRect();

virtual void update(float time) = 0;

virtual void checkCollisionWithMap(float dx, float dy) = 0;

virtual void interaction(Entity\* obj, float time) = 0;

};

**InitEntities.cpp**

#include "InitEntities.h"

void initEntities(Level\* lvl, std::list<Entity\*>& entities)

{

Image easyEnemyImage;

easyEnemyImage.loadFromFile("images/easyEnemy.png");

easyEnemyImage.createMaskFromColor(Color(255, 0, 0));

Image movePlatformImage;

movePlatformImage.loadFromFile("images/MovingPlatform.png");

Image medicalsImage;

medicalsImage.loadFromFile("images/Medicals.png");

Image moneyImage;

moneyImage.loadFromFile("images/Money.png");

Image Enemy\_2\_Image;

Enemy\_2\_Image.loadFromFile("images/Enemy2.png");

Image Enemy\_3\_Image;

Enemy\_3\_Image.loadFromFile("images/Enemy3.png");

Image checkpointImage;

checkpointImage.loadFromFile("images/Checkpoint.png");

Image lastCheckpointImage;

lastCheckpointImage.loadFromFile("images/LastCheckpoint.png");

//второй итератор.для взаимодействия между объектами списка

std::vector<Object> e = (\*lvl).GetObjects("EasyEnemy");

for (int i = 0; i < e.size(); i++) entities.push\_back(new Enemy(easyEnemyImage, "EasyEnemy", \*lvl, e[i].rect.left, e[i].rect.top, 50, 90, entities));

e = (\*lvl).GetObjects("MovingPlatform");//забираем все платформы в вектор

for (int i = 0; i < e.size(); i++) entities.push\_back(new MovingPlatform(movePlatformImage, "MovingPlatform", \*lvl, e[i].rect.left, e[i].rect.top, 95, 22));

e = (\*lvl).GetObjects("Enemy2");//забираем все платформы в вектор

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

entities.push\_back(new Enemy(Enemy\_2\_Image, "Enemy2", \*lvl, e[i].rect.left, e[i].rect.top, 101, 37, entities));

}

e = (\*lvl).GetObjects("Enemy3");//забираем все платформы в вектор

for (int i = 0; i < e.size(); i++) { entities.push\_back(new Enemy(Enemy\_3\_Image, "Enemy3", \*lvl, e[i].rect.left, e[i].rect.top, 59, 68, entities)); }

e = (\*lvl).GetObjects("Money");

for (int i = 0; i < e.size(); i++) { entities.push\_back(new Boost(moneyImage, "Money", \*lvl, e[i].rect.left, e[i].rect.top, 20, 20)); }

e = (\*lvl).GetObjects("Medicals");

for (int i = 0; i < e.size(); i++) entities.push\_back(new Boost(medicalsImage, "Medicals", \*lvl, e[i].rect.left, e[i].rect.top, 20, 20));

e = (\*lvl).GetObjects("Checkpoint");//забираем все платформы в вектор

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

entities.push\_back(new Checkpoint(checkpointImage, "Checkpoint", \*lvl, e[i].rect.left, e[i].rect.top, 88, 56));

}

e = (\*lvl).GetObjects("LastCheckpoint");//забираем все платформы в вектор

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

entities.push\_back(new Checkpoint(lastCheckpointImage, "LastCheckpoint", \*lvl, e[i].rect.left, e[i].rect.top, 88, 56));

}

}

**InitEntities.h**

#pragma once

#include "Level.h"

#include "View.h"

#include <list>

#include "Entity.h"

#include"Player.h"

#include "MovingPlatform.h"

#include"Enemy.h"

#include"Boost.h"

#include "Checkpoint.h"

void initEntities(Level\* lvl, std::list<Entity\*>& entities);

**Kursach(1.0).cpp**

#include <SFML/Graphics.hpp>

#include "Level.h"

#include <vector>

#include "iostream"

#include <list>

#include "View.h"

#include "Entity.h"

#include "MovingPlatform.h"

#include "Bullet.h"

#include "Enemy.h"

#include "Player.h"

#include "ObjectsInteraction.h"

#include "InitEntities.h"

#include "UpdateEntities.h"

#include <iostream>

#include "menu.h"

#include "Mission.h"

#include<string>

using namespace std;

using namespace sf;

int main()

{

RenderWindow window(VideoMode(1376, 768), "Shooter");

menu(window);

return 0;

}

**LifeBar.cpp**

#include "LifeBar.h"

LifeBar::LifeBar()

{

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

image.createMaskFromColor(Color(50, 96, 166));

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setTextureRect(IntRect(783, 2, 15, 84));

bar.setFillColor(Color(0, 0, 0));

max = 100;

}

void LifeBar::update(int health)

{

if (health > 0 && health <= max)

{

bar.setSize(Vector2f(10, (max - health) \* 70 / max));

}

else if (health == max)

{

bar.setSize(Vector2f(0, 0));

}

}

void LifeBar::draw(RenderWindow& window)

{

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

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

sprite.setPosition(center.x - size.x / 2 + 10, center.y - size.y / 2 + 10);

bar.setPosition(center.x - size.x / 2 + 14, center.y - size.y / 2 + 14);

window.draw(sprite);

window.draw(bar);

}

**LifeBar.h**

#pragma once

#include <SFML/Graphics.hpp>

#include <iostream>

using namespace sf;

class LifeBar

{

public:

Image image;

Texture texture;

Sprite sprite;

int max;

RectangleShape bar;

LifeBar();

void update(int health);

void draw(RenderWindow& window);

};

**LoadParams.h**

#pragma once

#include <string>

#include <list>

#include <SFML/Graphics.hpp>

using namespace std;

struct Lvl

{

bool complete;

string lvlPath;

string lvlName;

};

struct LoadParams

{

int score;

string bulletType;

list<Lvl> lvlList;

};

**Menu.h**

#include <SFML/Graphics.hpp>

#include "Play.h"

#include "LoadParams.h"

#include "SaveData.h"

#include <string>

#include <fstream>

#pragma once

using namespace std;

using namespace sf;

LoadParams params;

bool initParams()

{

ifstream file("params.txt");

if (file.is\_open())

{

string s;

while (getline(file, s))

{

if (s == "Score")

{

getline(file, s);

params.score = stoi(s);

}

if (s == "Level")

{

Lvl lvl;

getline(file, s);

lvl.lvlName = s;

getline(file, s);

lvl.lvlPath = s;

getline(file, s);

lvl.complete = stoi(s);

params.lvlList.push\_back(lvl);

}

}

params.bulletType = "Bullet";

file.close();

}

else {

return false;

}

}

void menu(RenderWindow& window) {

list<Lvl>\* levels = &params.lvlList;

list<Lvl>::iterator it;

Font font;

font.loadFromFile("CyrilicOld.ttf");

Text text("", font, 20);

text.setFillColor(Color::Red);

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

std::ostringstream scoreString; // объявили переменну

initParams();

Texture menuTexture1, menuTexture2, menuTexture3, aboutTexture, menuBackground;

menuTexture1.loadFromFile("images/newGameTexture.png");

menuTexture2.loadFromFile("images/chooseLevelTexture.png");

menuTexture3.loadFromFile("images/exitGameTexture.png");

//aboutTexture.loadFromFile("images/chooseLevelTexture.png");

menuBackground.loadFromFile("images/Dracula.jpg");

Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3), about(aboutTexture), menuBg(menuBackground);

bool isMenu = 1;

int menuNum = 0;

menu1.setPosition(100, 30);

menu2.setPosition(100, 90);

menu3.setPosition(100, 150);

menuBg.setPosition(345, 0);

//////////////////////////////МЕНЮ///////////////////

while (isMenu && window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

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

saveData(params);

window.close();

}

}

scoreString << params.score;

text.setString("Total score:" + scoreString.str());

scoreString.str("");

text.setPosition(window.getSize().x - 200, 50);

menu1.setColor(Color::White);

menu2.setColor(Color::White);

menu3.setColor(Color::White);

menuNum = 0;

window.clear(Color::Black);

if (IntRect(100, 30, 300, 50).contains(Mouse::getPosition(window))) { menu1.setColor(Color::Blue); menuNum = 1; }

if (IntRect(100, 90, 300, 50).contains(Mouse::getPosition(window))) { menu2.setColor(Color::Blue); menuNum = 2; }

if (IntRect(100, 150, 300, 50).contains(Mouse::getPosition(window))) { menu3.setColor(Color::Blue); menuNum = 3; }

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

{

if (menuNum == 1) {

params.score = 0;

for (it = levels->begin(); it != levels->end(); it++)

{

(\*it).complete = false;

}

saveData(params);

}//если нажали первую кнопку, то выходим из меню

if (menuNum == 2) {

play(params, window);

}

if (menuNum == 3) { saveData(params); window.close(); isMenu = false; }

}

window.setView(window.getDefaultView());

window.draw(menuBg);

window.draw(menu1);

window.draw(menu2);

window.draw(menu3);

window.draw(text);

window.display();

}

////////////////////////////////////////////////////

}

**Mission.cpp**

#include "Mission.h"

#include "LifeBar.h"

void Mission::start(RenderWindow& window)

{

LifeBar liefeBarPlayer;

std::list<Entity\*> entities;

Font font;

font.loadFromFile("CyrilicOld.ttf");

Text text("", font, 20);

text.setFillColor(Color::Red);

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

std::ostringstream scoreString; // объявили переменную

int numOfAct = 0;

bool isEsc = false;

bool isMission = true;

string lvlPath = params->lvlPath;

Object player;

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

Level lvl;

View view;

Image heroImage;

Image sightImage;

lvl.LoadFromFile(lvlPath);

player = lvl.GetObject("Player");

view.reset(FloatRect(0, 0, 640, 480));

heroImage.loadFromFile("images/player.png");

sightImage.loadFromFile("images/dot.png");

Player p(heroImage, sightImage, "Player", lvl, view, player.rect.left, player.rect.top, 40, 50, 100, 100);

Clock clock;

Texture yesTexture;

Texture noTexture;

yesTexture.loadFromFile("images/yes.png");

noTexture.loadFromFile("images/no.png");

Texture questionTexture;

questionTexture.loadFromFile("images/question.png");

Sprite question(questionTexture);

question.setOrigin(question.getTextureRect().width / 2, question.getTextureRect().height / 2);

Texture backgroundTexture;

backgroundTexture.loadFromFile("images/background.jpg");

Sprite background(backgroundTexture);

background.setOrigin(background.getTextureRect().width / 2, background.getTextureRect().height / 2);

Sprite yes(yesTexture);

yes.setOrigin(yes.getTextureRect().width / 2, yes.getTextureRect().height / 2);

Sprite no(noTexture);

no.setOrigin(no.getTextureRect().width / 2, no.getTextureRect().height / 2);

initEntities(&lvl, entities);

while (window.isOpen() && isMission)

{

numOfAct = 0;

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

clock.restart();

time = time / 800;

Event event;

while (window.pollEvent(event))

{

//второй варант стрельнуть

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

{

if (Keyboard::isKeyPressed(Keyboard::Escape)) {

isEsc = !isEsc;

if (isEsc)

{

background.setPosition(view.getCenter().x, view.getCenter().y);

question.setPosition(background.getPosition().x, background.getPosition().y - 50);

yes.setPosition(background.getPosition().x - 200, background.getPosition().y + 100);

no.setPosition(background.getPosition().x + 200, background.getPosition().y + 100);

}

}

if (event.key.code == sf::Keyboard::P && !Mp.isFired)

{

p.fire(entities);

}

}

}

liefeBarPlayer.update(p.health);

updateEntities(entities, time, p);

p.update(time);//перенесли сюда update игрока

for (it = entities.begin(); it != entities.end(); it++)//проходимся по эл-там списка

{

p.interaction(\*it, time);

(\*it)->interaction(&p, time);

}

if (!p.life)

{

Image medicalsImage;

medicalsImage.loadFromFile("images/Medicals.png");

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

for (int i = 0; i < e.size(); i++) entities.push\_back(new Boost(medicalsImage, "Medicals", lvl, e[i].rect.left, e[i].rect.top, 20, 20));

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

(\*it)->health = 100;

}

p.x = p.savedX;

p.y = p.savedY;

p.state = Player::State::stay;

p.health = 100;

p.life = true;

setPlayerCoordinateForView(p.savedX, p.savedY, \*p.view);

}

window.clear(Color(77, 83, 140));

lvl.Draw(window);

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

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

}

window.draw(p.sight);

window.draw(p.sprite);

scoreString << p.score;

text.setString("Score:" + scoreString.str());

scoreString.str("");

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

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

text.setPosition(center.x + size.x / 2 - 100, center.y - size.y / 2 + 10);

window.draw(text);

liefeBarPlayer.draw(window);

if (isEsc)

{

setPlayerCoordinateForView(background.getPosition().x, background.getPosition().y, view);

int x = window.mapCoordsToPixel(yes.getPosition()).x;

int y = window.mapCoordsToPixel(yes.getPosition()).y;

if (IntRect(x - yes.getTextureRect().width / 2, y - yes.getTextureRect().height / 2, yes.getTextureRect().width, yes.getTextureRect().height).contains(Mouse::getPosition(window)))

{

numOfAct = 1;

}

x = window.mapCoordsToPixel(no.getPosition()).x;

y = window.mapCoordsToPixel(no.getPosition()).y;

if (IntRect(x - no.getTextureRect().width / 2, y - no.getTextureRect().height / 2, no.getTextureRect().width, no.getTextureRect().height).contains(Mouse::getPosition(window)))

{

numOfAct = 2;

}

window.draw(background);

window.draw(question);

window.draw(yes);

window.draw(no);

}

window.setView(view);

window.display();

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

{

if (numOfAct == 1)

{

isMission = false;

}

if (numOfAct == 2)

{

isEsc = false;

}

}

if (p.missionComplete)

{

params->isComplete = true;

params->score += p.score;

isMission = false;

}

}

for (it = (entities).begin(); it != (entities).end(); )//говорим что проходимся от начала до конца

{

delete\* it;

it = (entities).erase(it);// delete (\*it);

}

}

Mission::Mission(Params& params)

{

this->params = &params;

}

**Mission.h**

#pragma once

#include <SFML/Graphics.hpp>

#include "Level.h"

#include <vector>

#include "iostream"

#include <list>

#include "View.h"

#include "Entity.h"

#include "MovingPlatform.h"

#include "Bullet.h"

#include "Enemy.h"

#include "Player.h"

#include "ObjectsInteraction.h"

#include "InitEntities.h"

#include "UpdateEntities.h"

#include <iostream>

#include "Mission.h"

#include<string>

#include"Params.h"

#include<sstream>

using namespace sf;

using namespace std;

class Mission

{

public:

Params\* params;

Mission(Params& params);

void start(RenderWindow& window);

};

**MovingPlatform.cpp**

#include "MovingPlatform.h"

MovingPlatform::MovingPlatform(Image& image, String name, Level& lvl, float x, float y, int w, int h) :Entity(image, name, x, y, h, w, 0) {

sprite.setTextureRect(IntRect(0, 0, w, h));//прямоугольник

dx = 0.08;//изначальное ускорение по Х

}

void MovingPlatform::update(float time)//функция обновления платформы.

{

x += dx \* time;//реализация движения по горизонтали

moveTimer += time;//наращиваем таймер

if (moveTimer > 2000) { dx \*= -1; moveTimer = 0; }//если прошло примерно 2 сек, то меняется направление движения платформы, а таймер обнуляется

sprite.setPosition(x + w / 2, y + h / 2);//задаем позицию спрайту

}

void MovingPlatform::interaction(Entity\* obj, float time)

{

}

void MovingPlatform::checkCollisionWithMap(float dx, float dy)

{

}

**MovingPlatform.h**

#pragma once

#include "Entity.h"

class MovingPlatform :

public Entity

{

public:

MovingPlatform(Image& image, String name, Level& lvl, float x, float y, int w, int h);

void update(float time);

void interaction(Entity\* obj, float time);

void checkCollisionWithMap(float dx, float dy);

};

**ObjectInteraction.cpp**

#include"ObjectsInteraction.h"

void interact(float time, std::list<Entity\*>& entities)

{

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

std::list<Entity\*>::iterator it2;

for (it = entities.begin(); it != entities.end(); it++)//проходимся по эл-там списка

{

for (it2 = entities.begin(); it2 != entities.end(); it2++)

{

if ((\*it)->getRect() != (\*it2)->getRect())//при этом это должны быть разные прямоугольники

(\*it)->interaction(\*it2, time);

}

}

}

**ObjectInteraction.h**

#pragma once

#include "Entity.h"

#include <list>

#include "Player.h"

void interact(float time, std::list<Entity\*>& entities);

**Params.h**

#pragma once

#include <string>

using namespace std;

struct Params

{

int score;

string lvlPath;

bool isComplete;

string bulletType;

};

**Play.h**

#include "LoadParams.h"

#include"Mission.h"

#include "SaveData.h"

using namespace std;

#pragma once

bool isPlay = false;

void play(LoadParams& loadParams, RenderWindow& window)

{

int lvlNum;

isPlay = true;

list<Lvl>\* levels = &loadParams.lvlList;

list<Lvl>::iterator it;

Lvl\* prev;

prev = &(\*levels->begin());

Texture lvlTexture1;

Texture lvlTexture2;

Texture lvlTexture3;

Texture backgroundTexture;

Texture playTexture;

Texture latchTexture;

lvlTexture1.loadFromFile("images/lvlTexture1.png");

lvlTexture2.loadFromFile("images/lvlTexture2.png");

lvlTexture3.loadFromFile("images/lvlTexture3.png");

backgroundTexture.loadFromFile("images/backgroundTexture.jpg");

int x = 250;

int y = 300;

int w = 69;

int h = 48;

Sprite level1(lvlTexture1);

Sprite level2(lvlTexture2);

Sprite level3(lvlTexture3);

Sprite background(backgroundTexture);

for (it = levels->begin(); it != levels->end(); it++)

{

if ((\*it).lvlName == "L1")

{

level1.setPosition(x, y);

}

else if ((\*it).lvlName == "L2")

{

level2.setPosition(x, y);

}

else if ((\*it).lvlName == "L3")

{

level3.setPosition(x, y);

}

x += w + 250;

if ((\*it).complete == true)

{

}

else

{

}

}

while (window.isOpen() && isPlay)

{

window.clear(Color(129, 181, 221));

level1.setColor(Color::White);

level2.setColor(Color::White);

level3.setColor(Color::White);

lvlNum = 0;

Event event;

while (window.pollEvent(event))

{

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

{

saveData(loadParams);

window.close();

}

}

for (it = levels->begin(); it != levels->end(); it++)

{

Sprite\* s = &level1; ;

if (prev->complete == true)

{

if ((\*it).lvlName == "L2")

{

s = &level2;

}

if ((\*it).lvlName == "L3")

{

s = &level3;

}

}

if (IntRect(s->getPosition().x, s->getPosition().y, s->getTextureRect().width, s->getTextureRect().height).contains(Mouse::getPosition(window)))

{

s->setColor(Color::Blue);

if (s == &level1)

{

lvlNum = 1;

}

if (s == &level2)

{

lvlNum = 2;

}

if (s == &level3)

{

lvlNum = 3;

}

}

prev = &(\*it);

}

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

{

if (lvlNum == 1) {//если нажали первую кнопку, то выходим из меню

for (it = levels->begin(); it != levels->end(); it++)

{

if ((\*it).lvlName == "L1")

{

Params params = { loadParams.score,it->lvlPath,it->complete,"Bullet" };

Mission mission(params);

mission.start(window);

it->complete = params.isComplete;

loadParams.score = params.score;

isPlay = false;

}

}

}

if (lvlNum == 2) {

for (it = levels->begin(); it != levels->end(); it++)

{

if ((\*it).lvlName == "L2")

{

Params params = { loadParams.score,it->lvlPath,it->complete,"Bullet" };

Mission mission(params);

mission.start(window);

it->complete = params.isComplete;

loadParams.score = params.score;

isPlay = false;

}

}

}

if (lvlNum == 3) {

for (it = levels->begin(); it != levels->end(); it++)

{

if ((\*it).lvlName == "L2")

{

Params params = { loadParams.score,it->lvlPath,it->complete,"Bullet" };

Mission mission(params);

mission.start(window);

it->complete = params.isComplete;

loadParams.score = params.score;

isPlay = false;

}

}

}

}

window.draw(background);

window.draw(level1);

window.draw(level2);

window.draw(level3);

window.display();

}

}

**Player.cpp**

#include "Player.h"

#include <iostream>

using namespace std;

Player::Player(Image& image, Image& sightImage, String name, Level& level, View& view, float x, float y, int w, int h, int sW, int sH) :Entity(image, name, x, y, h, w, 0)

{

fireTimer = 500;

isFalling = false;

isClimbed = false;

isDamaged = false;

isFired = false;

damageTimer = 3000;

this->health = 100;

missionComplete = false;

onStair = false;

onGround = false;

savedX = x;

savedY = y;

this->sW = sW;

this->sH = sH;

this->level = level;

this->sightImage = sightImage;

isShoot = isAimed = 0; state = stay; obj = level.GetAllObjects(); this->view = &view; rad = 35; aimDeg = 0;

dirSX = cos((aimDeg / 180.0) \* acos(-1));

dirSY = sin((aimDeg / 180.0) \* acos(-1));

this->sX = this->x + rad \* dirSX + w / 2;

this->sY = this->y + rad \* dirSY + h / 2;

if (name == "Player") {

sprite.setTextureRect(IntRect(0, 0, w, h));

}

this->texture.loadFromImage(sightImage);

sight.setTexture(texture);

sight.setOrigin(sW / 2, sH / 2);

sight.setTextureRect(IntRect(950, 950, sW, sH));

sight.setScale(0.1, 0.1);

this->sW \*= 0.1;

this->sH \*= 0.1;

}

void Player::control() {

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

state = right; speed = 0.1; isMove = true;

}

else

{

isMove = false;

}

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

state = left; speed = 0.1; isMove = true;

}

else

{

isMove = false;

}

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

state = jump; dy = -0.6; onGround = false;

}

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

state = up; speed = 0.1; onGround = false; isClimbed = true;

}

else if (Keyboard::isKeyPressed(Keyboard::Down) && (onStair)) {

state = down; speed = 0.1; isClimbed = true;

}

else

{

isClimbed = false;

}

/////выстрел

if (Keyboard::isKeyPressed(Keyboard::Space)) {

isShoot = true;

}

if (Keyboard::isKeyPressed(Keyboard::W)) {

aimSpeed = 0.05;

state = aimUp; isAimed = true;

}

else if (Keyboard::isKeyPressed(Keyboard::S)) {

aimSpeed = 0.05;

state = aimDown; isAimed = true;

}

else

{

isAimed = false;

}

}

void Player::checkCollisionWithMap(float dx, float dy)

{

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

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

{

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

{

if (dy > 0) { y = obj[i].rect.top - h; this->dy = 0; onGround = true; }

if (dy < 0) { y = obj[i].rect.top + obj[i].rect.height; this->dy = 0; }

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

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

}

}

}

}

void Player::update(float time)

{

control();

switch (state)

{

case right:dx = speed; break;

case left:dx = -speed; break;

case up:dy = -speed; break;

case down: dy = speed; break;

case stay: break;

case right\_Top:dx = speed; break;//состояние вправо вверх, просто продолжаем идти вправо

}

switch (state)

{

case aimUp:dAim = -aimSpeed; break;

case aimDown:dAim = aimSpeed; break;

}

x += dx \* time;

checkCollisionWithMap(dx, 0);

y += dy \* time;

checkCollisionWithMap(0, dy);

aimDeg = aimDeg + dAim \* time;

if (aimDeg < -90)

{

aimDeg = -90;

}

else if (aimDeg > 90)

{

aimDeg = 90;

}

if ((state == right) || (dirSX > 0 && state == stay))

{

dirSX = cos((aimDeg / 180.0) \* acos(-1));

}

else if ((state == left) || (dirSX < 0 && state == stay))

{

dirSX = -cos((aimDeg / 180.0) \* acos(-1));

}

else if (dirSX > 0)

{

dirSX = cos((aimDeg / 180.0) \* acos(-1));

}

else if (dirSX < 0)

{

dirSX = -cos((aimDeg / 180.0) \* acos(-1));

}

dirSY = sin((aimDeg / 180.0) \* acos(-1));

sX = x + w / 2 + rad \* dirSX;

sY = y + h / 2 + rad \* dirSY;

//cout << sX << " " << sY << " " << x << " " << y << endl;

sight.setPosition(sX + sW / 2, sY + sH / 2);

sprite.setPosition(x + w / 2, y + h / 2);

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

if (!isAimed) {

dAim = 0;

aimSpeed = 0;

}

if (!isMove) {

speed = 0;

dx = 0;

}

if (life) {

setPlayerCoordinateForView(x, y, \*view);

}

//cout << onGround << " " << onStair << endl;

if (!onStair && !isFalling)

{

dy = speed;

isFalling = true;

isClimbed = false;

}

if (onGround || onStair)

{

isFalling == false;

}

dy = dy + 0.0015 \* time;

if (!isClimbed && onStair)

{

dy = 0;

}

if (isDamaged)

{

if (damageTimer > 0)

{

sprite.setColor(Color::Red);

damageTimer -= time;

}

else

{

sprite.setColor(Color::White);

isDamaged = false;

damageTimer = 3000;

}

}

onStair = false;

if (isFired)

{

if (fireTimer > 0)

{

fireTimer -= time;

}

else

{

isFired = false;

fireTimer = 500;

}

}

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

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

{

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

{

if (!isDamaged) {

isDamaged = true;

health -= 20;

}

}

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

{

onStair = true;

}

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

{

if (!isDamaged) {

isDamaged = true;

health -= 20;

}

}

} // else { onGround = false; } }

}

if (state == right)

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

else if (state == left) {

sprite.setTextureRect(IntRect(w, 0, -w, h));

}

}

void Player::interaction(Entity\* obj, float time)

{

if ((obj->name == "MovingPlatform") && (obj->getRect().intersects(this->getRect())))//если игрок столкнулся с объектом списка и имя этого объекта movingplatform

{

Entity\* movPlat = obj;

if ((this->dy > 0) || (this->onGround == false))//при этом игрок находится в состоянии после прыжка, т.е падает вниз

if (this->y + this->h < movPlat->y + movPlat->h)//если игрок находится выше платформы, т.е это его ноги минимум (тк мы уже проверяли что он столкнулся с платформой)

{

this->y = movPlat->y - this->h + 3; this->x += movPlat->dx \* time; this->dy = 0; this->onGround = true; // то выталкиваем игрока так, чтобы он как бы стоял на платформе

}

}

if ((obj->name == "EasyEnemy" || obj->name == "Enemy2" || obj->name == "Enemy3") && (obj->getRect().intersects(this->getRect())))

{

if (!isDamaged) {

isDamaged = true;

health -= 20;

}

}

if ((obj->name == "Money") && (obj->getRect().intersects(this->getRect())))

{

}

if ((obj->name == "Medicals") && (obj->getRect().intersects(this->getRect())))

{

if (health != 100)

{

health += 10;

}

}

if ((obj->name == "Checkpoint") && (obj->getRect().intersects(this->getRect())))

{

savedX = obj->x;

savedY = obj->y;

obj->life = false;

cout << "." << endl;

}

if ((obj->name == "LastCheckpoint") && (obj->getRect().intersects(this->getRect())))

{

missionComplete = true;

obj->life = false;

}

if ((obj->name == "EBullet") && (obj->getRect().intersects(this->getRect())))

{

if (!isDamaged) {

isDamaged = true;

}

health -= 5;

}

if ((obj->name == "Stair") && (obj->getRect().intersects(this->getRect())))

{

}

}

void Player::fire(std::list<Entity\*>& entities)

{

isFired = true;

Image BulletImage;//изображение для пули

BulletImage.loadFromFile("images/bullet.png");//загрузили картинку в объект изображения

BulletImage.createMaskFromColor(Color(0, 0, 0));//маска для пули по черному цвету

this->isShoot = false;

(entities).push\_back(new Bullet(BulletImage, "Bullet", level, sX, sY, 16, 16, 0.8, dirSX, dirSY));

}

**Player.h**

#pragma once

#include "Entity.h"

#include "view.h"

#include <list>

#include "Bullet.h"

#include<math.h>

class Player :

public Entity

{

public:

enum State { left, right, up, down, jump, stay, right\_Top, aimUp, aimDown };

State state;

View\* view;

float fireTimer;

bool isFired;

int damageTimer;

float aimDeg;

float aimSpeed;

float dAim;

float rad;

bool isDamaged;

bool isFalling;

bool missionComplete;

bool isAimed;

bool isShoot;

bool onStair;

bool isClimbed;

Level level;

Image sightImage;

Texture texture;

Sprite sight;

float dirSX, dirSY, sX, sW, sH, sY, savedX, savedY;

Player(Image& image, Image& sightImage, String name, Level& level, View& view, float x, float y, int w, int h, int sW, int sH);

void control();

void checkCollisionWithMap(float dx, float dy);

void update(float time);

void interaction(Entity\* obj, float time);

void fire(std::list<Entity\*>& entities);

};

**SaveData.h**

#pragma once

#include "LoadParams.h"

#include <string>

#include <fstream>

void saveData(LoadParams& params)

{

ofstream file("params.txt");

string s;

file << "Score" << endl;

file << params.score << endl;

list<Lvl>\* levels = &params.lvlList;

list<Lvl>::iterator it;

for (it = levels->begin(); it != levels->end(); it++)

{

file << "Level" << endl;

file << it->lvlName << endl;

file << it->lvlPath << endl;

file << it->complete << endl;

}

file.close();

}

**UpdateEntities.cpp**

#include "UpdateEntities.h"

#include <iostream>

using namespace std;

void updateEntities(std::list<Entity\*>& entities, float time, Player& p)

{

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

for (it = entities.begin(); it != entities.end();)//говорим что проходимся от начала до конца

{

Entity\* b = \*it;//для удобства, чтобы не писать (\*it)->

b->update(time);//вызываем ф-цию update для всех объектов (по сути для тех, кто жив)

if (b->life == false) {

p.score += b->score;

it = (entities).erase(it); delete b;

}// если этот объект мертв, то удаляем его

else it++;//и идем курсором (итератором) к след объекту. так делаем со всеми объектами списка

}

interact(time, (entities));

}

**UpdateEntities.h**

#pragma once

#include "ObjectsInteraction.h"

void updateEntities(std::list<Entity\*>& entities, float time, Player& p);

**View.cpp**

#include "View.h"

#include <iostream>

using namespace std;

void setPlayerCoordinateForView(float x, float y, View& view) {

float tempX = x; float tempY = y;

view.setCenter(tempX, tempY);

}

**View.h**

#pragma once

#include <SFML/Graphics.hpp>

using namespace sf;

void setPlayerCoordinateForView(float x, float y,View& view);

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 012 ПЗ | | | | Пояснительная записка | | | | 62с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951008  012 ПД | | | | Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 012 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Игровое программное средство “Shooter”  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Костюкевич П.Ю. |  |  | Т |  | |  | 62 | 62 |
| Пров. | | Марина И.М. |  |  | Кафедра ПОИТ  гр. 951008 | | | | | |
|  | |  |  |  |