МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

**Кафедра современных технологий программирования**

БЛОЦКИЙ ДМИТРИЙ СЕРГЕЕВИЧ

**Игровое приложение «Монетка»**

Курсовая работа

студента 1 курса специальности

1-40 01 01 «Программное обеспечение информационных технологий»

дневной формы получения образования

|  |  |
| --- | --- |
|  | Научный руководитель  Банюкевич Елена Викторовна,  Старший преподаватель кафедры современных технологий программирования. |
|  |  |

Гродно, 2019

# РЕЗЮМЕ

Тема курсовой работы

**игра «Монетка»**

Работа содержит: 31 страницы, 3 приложения, 5 использованных источников литературы.

Цель курсовой работы: разработка игрового приложения «Монетка»

Объект исследования: приложение «Монетка»

Предмет исследования: реализация приложения «Монетка» на ЭВМ с помощью языка программирования C++, с использованием SFML библиотеки.

Авторская характеристика работы: приложение «Монетка» написано на языке программирования C++, с использованием SFML библиотеки, и представляет собой простую реализацию, разработанную программой Microsoft Visual Studio.

# SUMMARY

Coursework topic

game «Coin»

The work contains: 31 pages, 3 applications, 5 used sources of literature.

The purpose of the course work: the development of the game application "Coin"

The object of study: the application "Coin"

Subject of research: the implementation of the application "Coin" on a computer using the programming language C ++, using SFML library.

Author's description of the work: the application “Coin” is written in the C ++ programming language, using the SFML library, and is a simple implementation developed by the Microsoft Visual Studio program.

СОДЕРЖАНИЕ

[РЕЗЮМЕ 2](#_Toc9894682)

[SUMMARY 3](#_Toc9894683)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ 5](#_Toc9894684)

[ВВЕДЕНИЕ 6](#_Toc9894685)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 8](#_Toc9894686)

[1.1. Основные аспекты 8](#_Toc9894687)

[1.2. Обзор существующих решений 9](#_Toc9894688)

[1.3. Выводы по главе 1 11](#_Toc9894689)

[2. ПРОЕКТИРОВАНИЕ ИГРЫ «МОНЕТКА» 12](#_Toc9894690)

[2.1. Этапы разработки приложения. 12](#_Toc9894691)

[2.2. Функциональное моделирование. 12](#_Toc9894692)

[2.3. Проектирование графического интерфейса программы. 14](#_Toc9894693)

[2.4. Выводы по главе 2. 15](#_Toc9894694)

[3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «МОНЕТКА» 16](#_Toc9894695)

[3.1. Основные элементы 16](#_Toc9894696)

[3.2. Основная часть 17](#_Toc9894697)

[3.2.1. Class Entity 17](#_Toc9894698)

[3.2.2. Class Player 18](#_Toc9894699)

[3.2.3. Работа с камерой вида 20](#_Toc9894700)

[3.2.4. Взаимодействие с картой 22](#_Toc9894701)

[3.2.5. Работа с текстом 23](#_Toc9894702)

[3.3 Выводы по главе 3 24](#_Toc9894703)

[ЗАКЛЮЧЕНИЕ 25](#_Toc9894704)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc9894705)

[ПРИЛОЖЕНИЕ 1 28](#_Toc9894706)

[ПРИЛОЖЕНИЕ 2 29](#_Toc9894707)

[ПРИЛОЖЕНИЕ 3 31](#_Toc9894708)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ

|  |  |  |
| --- | --- | --- |
| SFML  ООП  Sprite  Tile | Simple and Fast Multimedia Library  Объектно-ориентированное программирование  Спрайт  Плитка | Простая и быстрая мультимедийная библиотека, представляющая собой объектно-ориентированную обёртку для OpenGL.  Методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.  Графический объект в компьютерной графике.  Небольшие изображения одинаковых размеров, которые и служат фрагментами большой картины. |

# ВВЕДЕНИЕ

Игры в современном обществе занимают лидирующее место в сфере развлечений. Уже давно многие представляли игровую индустрию как отдельный вид искусства. Что впоследствии и случилось. В 2011 году компьютерные игры официально признаны как отдельный вид искусства в США. С каждым годом игровая индустрия развивается, соответственно, расширяя свои возможности. В настоящее время разработка игры - это многомиллионный процесс, в котором задействована целая команда разработчиков, сложные современные технологии и даже маркетинговые ходы. Так же существует множество книг и справочников о том «Как же начать писать игры?». В них пошагово объясняется и показывается, как начать создавать игровые приложения от самых простых до сложных с уникальным дизайном и простейшими методами и алгоритмами написания.

**Актуальность курсовой работы** заключается в развитии быстроты реакции, мышления и внимания.

**Главная задача** курсовой работы заключается в правильности составленных алгоритмов. Потребуется создать полноценно-функционирующую игру в среде C++ и самостоятельно изучить некоторые функции и аспекты.

**Объект исследования:** приложение в жанре платформер.

**Предмет исследования:** реализация приложения «Монетка» на ЭВМ с помощью языка программирования C++, с использованием SFML библиотеки.

**Цель исследования:** разработка игрового приложения «Монетка» в жанре платформера, его непосредственная реализация и тестирование. В соответствии с объектом, предметом и целью исследования были поставлены следующие задачи:

* Изучить библиотеку SFML.
* разработать программу для создания игрового приложения при помощи языка программирования C++, с использованием стандартных библиотек; разработать пользовательский интерфейс позволяющий пользователю: безошибочно выполнять действия на ПК, использовать набор иерархических меню, систему подсказок, вернуться к предыдущему состоянию системы.
* Применение теоретических знаний по дисциплине «Основы алгоритмизации и программирования» и по темам, вынесенным на самостоятельное изучение, на практике для создания программного проекта, реализующего приложение научного характера.
* Получение необходимых практических навыков программирования.
* Приобретение навыков работы с SFML библиотекой и использование её в научных проектах.

**Методы исследования:** объектно-ориентированное программирование, SFML библиотека.

**Практическая значимость** курсовой работы определяется огромной популярностью игр в современном мире.

Курсовая работа состоит из двух частей: основной и специальной. Основная часть состоит из компонентов, используемых при написании программы, также методов реализации проекта. Специальная часть состоит из описания программы и рабочего процесса. Завершает курсовую работу заключение и список используемой литературы, которая была использована при составлении курсовой работы и программы.

**Общий объём курсовой работы** составляет 31 страницы. Список использованной литературы состоит из 5 источников.

# **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## Основные аспекты

Платформер – жанр [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Некоторые предметы, называемые пауэр-апами, наделяют управляемого игроком персонажа особой силой, которая обычно иссякает со временем (к примеру: силовое поле, ускорение, увеличение высоты прыжков). Коллекционные предметы, оружие и «пауэер-ап» собираются обычно простым прикосновением персонажа, и для применения не требуют специальных действий со стороны игрока. На рисунке 1.1 изображено игровое поле.



Рис. 1.1

Реже предметы собираются в «инвентарь» героя и применяются специальной командой (такое поведение более характерно для [аркадных головоломок](https://ru.wikipedia.org/w/index.php?title=%D0%90%D1%80%D0%BA%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F_%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BB%D0%BE%D0%BC%D0%BA%D0%B0&action=edit&redlink=1)). Сходный жанр компьютерных игр [сайд-скроллер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D0%B4-%D1%81%D0%BA%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80).

Противники (называемые «врагами») всегда многочисленные и разнородные, обладают примитивным [искусственным интеллектом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82), стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Иногда противник может быть нейтрализован либо прыжком ему на голову, либо из оружия, если им обладает герой. Смерть живых существ обычно изображается упрощённо или символически (существо исчезает или проваливается вниз за пределы экрана).

## 1.2. Обзор существующих решений

Программы-аналоги созданные рание:

* Terraria — компьютерная инди-игра в жанре приключенческой песочницы, разработанная американской студией Re-Logic. Terraria получила положительные оценки игровой прессы. Рецензенты похвалили реиграбельность, большое количество предметов и существ, делающее исследование игрового мира интереснее. Кроме того, они одобрили двумерную графику, сходную с компьютерными играми времён 16-разрядной игровой приставки SNES. В версии игры для Microsoft Windows нет режима обучения — рецензенты, обозревавшие первую версию Terraria, сочли это за недостаток, но потом разработчики ввели режим обучения в версиях для игровых приставок и мобильных устройств.
* Machinarium — компьютерная игра в жанре квеста, разработанная чешской студией Amanita Design. Наиболее радикальным отличием от других игр жанра является то, что взаимодействовать можно только с теми объектами, которые находятся в зоне досягаемости персонажа, а также то, что можно растягивать или ужимать тело протагониста, что вносит свои коррективы в «традиционность». Игра не содержит привычных диалогов, используя вместо них систему анимированных «облачков мыслей».
* Super Mario Bros.— видеоигра в жанре платформера, разработанная и выпущенная в 1985 году японской компанией Nintendo для платформы Famicom. Занесена в «Книгу рекордов Гиннесса» как самая продаваемая игра в истории. Марио стал символом компании Nintendo и одним из самых известных игровых персонажей в мире.

На сайте The Cutting Room Floor нашли баг, из-за которого Лакиту действовал не как надо, вместе с ссылкой на файл, исправляющий ошибку.

## 1.3. Выводы по главе 1

Рассмотрен жанр компьютерных игр платформер как игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Приведены аналоги таких приложений с указанием их достоинств и недостатков. Так, в приложении Terraria используется большое количество предметов и существ, делающее исследование игрового мира интереснее, в то же время в изначальной версии игры для Microsoft Windows не было режима обучения. Приложение Machinarium имеет радикальное отличие от других жанров игр и является оригинальным. Приложение Super Mario Bros. являлось самым продаваемым, и в то же время был обнаружен баг.

Таким образом, для создания современной игры требуется усвоить огромное количество информации из самых разных областей знаний.

# ПРОЕКТИРОВАНИЕ ИГРЫ «МОНЕТКА»

## 2.1. Этапы разработки приложения.

Для достижения поставленной цели необходимо:

изучить теоретические сведения, требуемые для решения поставленной цели;

изучить требования, которым должно удовлетворять приложение;

изучены аналогичные приложения;

C++, SFML библиотека для разработки приложения;

разработать удобный пользовательский интерфейс;

создать приложение, реализующее применение теоретических знаний на практике для создания программного проекта;

протестировать созданное приложение, при необходимости внести соответствующие изменения.

проектирование функций для работы с картой, меню и другими элементами.

## 2.2. Функциональное моделирование.

Контроль персонажем. Изображено на рисунке 2.1.

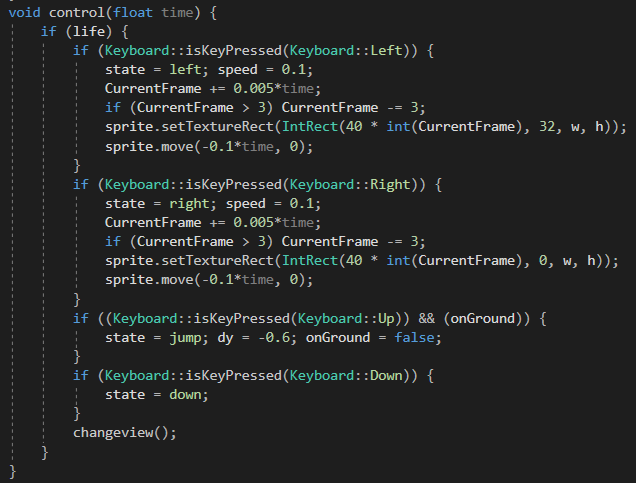


Рис. 2.1

Управление персонажем. Изображено на рисунке 2.2.

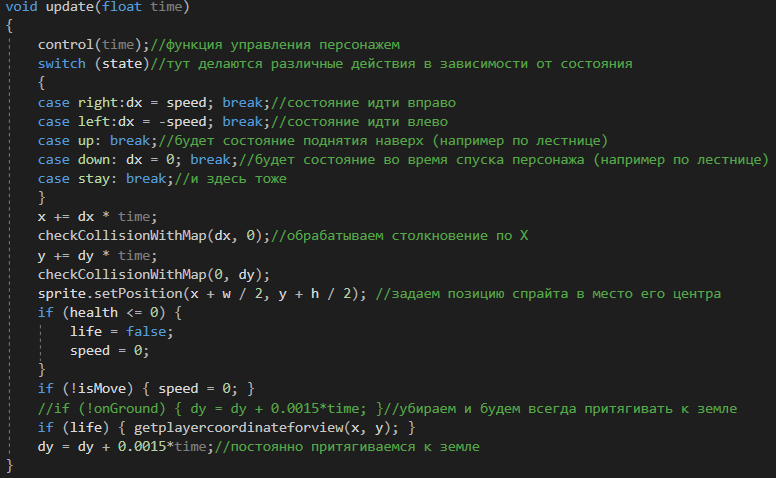


Рис. 2.2

Взаимодействие персонажа с картой. Изображено на рисунке 2.3.

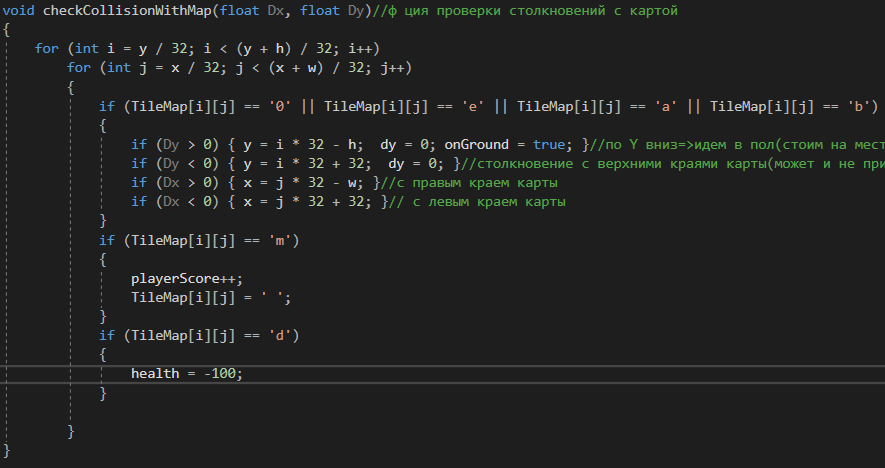


Рис. 2.3

## 2.3. Проектирование графического интерфейса программы.

Меню игры. . Изображено на рисунке 2.4.



Рис. 2.4

Игровое поле. Изображено на рисунке 2.5.

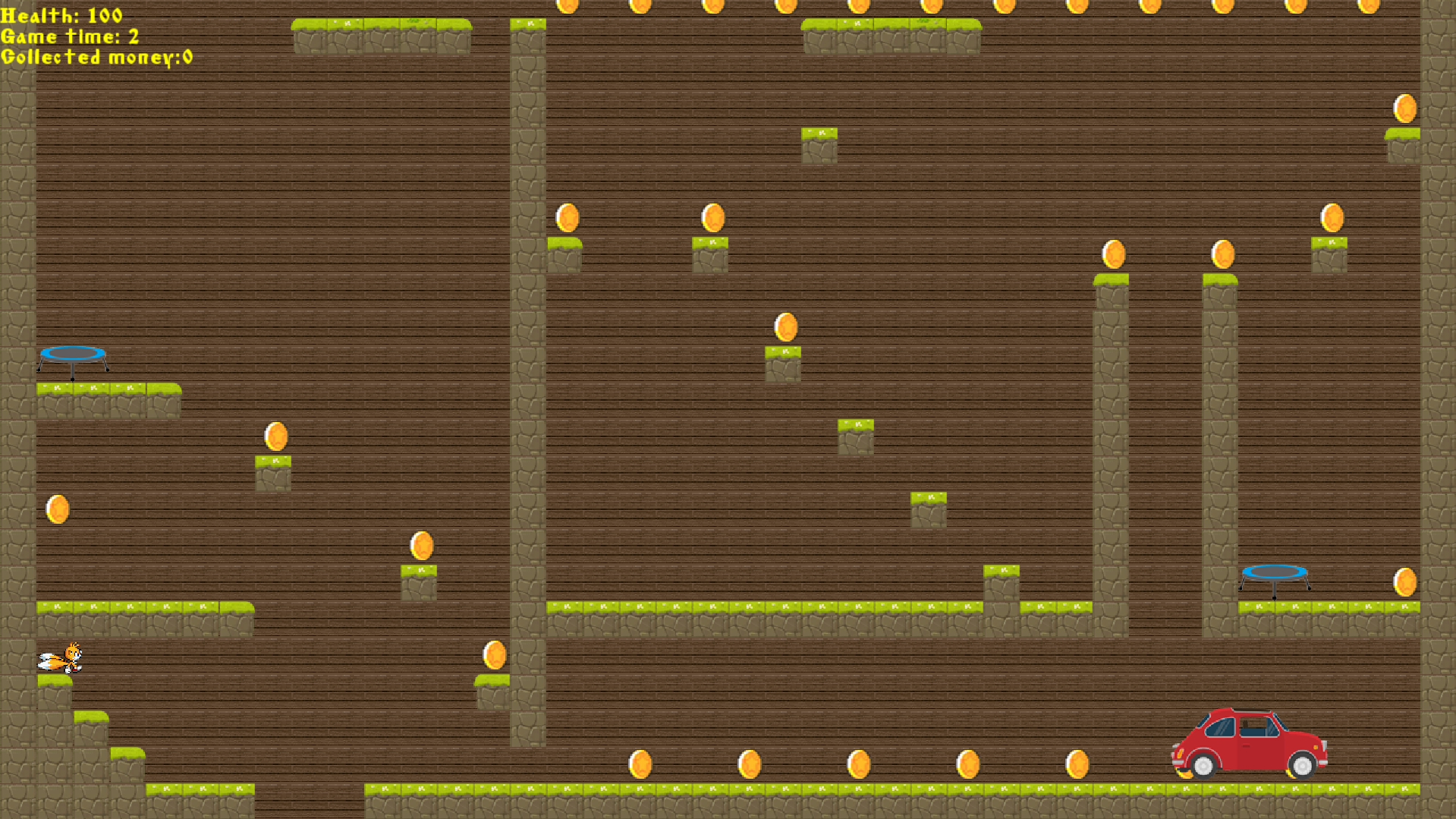


Рис. 2.5

## 2.4. Выводы по главе 2.

Исходя из всего вышеперечисленного и рассмотренного, можно понять, что создание игровых приложений является очень сложным, с одной стороны, но и очень увлекательным и развивающим процессом, с другой.

Игра является достаточно сложной, в своём роде, в плане поиска решений, но и, как оказалось, достаточно простой в реализации и довольно занимательной.

Не смотря на всё это, каждая программа или же игра имеет свои достоинства и недостатки, которые обуславливаются целевой аудиторией. Каждый игрок в процессе игры замечает то, что, по его мнению, можно было добавить как необходимую часть для игры, и то, что совсем не обязательно, и можно было исключить из кода игры. Кому-то нужен более простой интерфейс для удобства игры, а кто-то хочет более сложный для множества возможностей и функций игры.

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «МОНЕТКА»

## 3.1. Основные элементы

Основным и, пожалуй, самым первым элементом игрового приложения является карта, а если быть точнее, то её создание. Пользуясь широким функционалом и возможностями массива, создаётся карта “40x25”, в которую помещаются несколько символов = ‘ ‘ , ‘0’ , ‘m’ , то есть пробел, ноль, m. Когда рисуется карта, нужно спрашивать: если элемент массива равен ‘ ‘, то рисуется первый квадратик, если ‘m’, то рисуется монетка и т.д. На рисунке 2.1 изображен массив игрового поля.

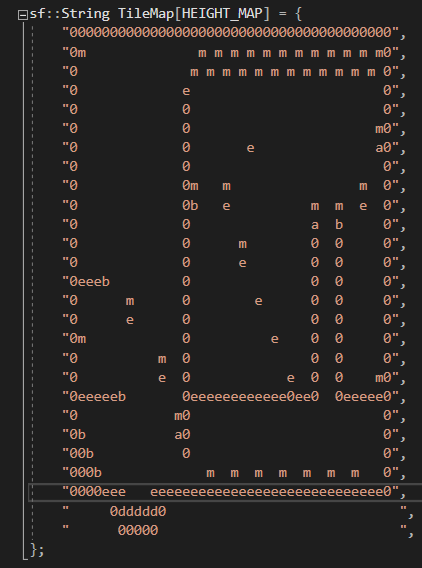


Рис. 3.1

## 3.2. Основная часть

## 3.2.1. Class Entity

В основу всей программы входит класс Entity class Entity, который включает в себя, в первую очередь, необходимые переменные размеров, объявление двумерного массива карты, а так же координат и многое другое.

Что общего между врагом и игроком? Очень многое: жизнь/смерть, здоровье, координаты у обоих есть, ширина, высота, функция обновления (update), столкновение с картой и многое-многое другое. Не нужно создавать класс врага и прописывать всё то же самое для него, добавляя ещё больше лишних строк кода. Да, этого не требуется, и нужно будет использовать наследование. Наследование дает те же переменные и функции классу потомку и можно будет их использовать, не описывая при этом по-новому, по сути, то же самое. При этом возможно переопределять те же функции, если функции родителя не устраивают. Общие переменные в public. Все они будут доступны для других дочерних классов, наследуемых от Entity. На рисунке 2.2 изображено наследование класса.

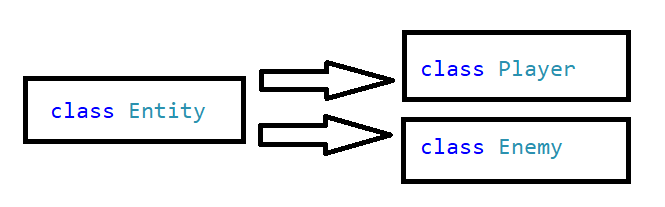


Рис. 3.2

Важный момент с изображением – здесь передается значение по ссылке. В функции main можно создавать изображения, например вот так:

Image heroImage;

heroImage.loadFromFile("MilesTailsPrower.gif");

И потом заряжать их в конструктор дочернего класса. В конструкторе дочернего класса оно передастся в конструктор класса родителя, а уже в конструкторе класса родителя (то есть Entity) оно закидывается в текстуру, затем в спрайт. Если бы можно было передавать не по ссылке, каждый раз при создании нового врага создавалась бы копия этого изображения, что раздувало бы память. В данном случае, получается, что нужно создавать изображение всего 1 раз.

Далее в конструкторе Entity принимаются такие параметры как: координаты появления объекта, ширина и высота объекта, и имя. Зачем нужно имя? Так возможно создавать различных врагов с разными изображениями и поведением и все они будут экземплярами класса Enemy. То есть не нужно делать отдельно класс HardEnemy, EasyEnemy, BossEnemy, и тд. Даётся имя врагу и в зависимости от его имени реализуется внутри класса логика для этого врага. Аналогично с игроком – может быть два игрока и два имени – “Player1″ и “Player2″. В общем, после всех этих параметров в конструкторе идет инициализация переменных. Эти переменные будут инициализированы сразу для всех дочерних классов при создании их экземпляров.

## 3.2.2. Class Player

Class Player исходит из класса Entity и поэтому всего две переменные имеет Игрок, этих переменных нет у других. Это очки игрока и состояние игрока.

В параметрах конструктора игрока имеется все то же изображение (именно то, которое загрузилось один раз в bool startGame), координаты и размеры будущего спрайта. Затем они передаются в базовый класс (класс родитель) Entity. Там уже эти переменные инициализируются, и изображение запиливается в спрайт.

В самом конструкторе, внутри для первого игрока делается прямоугольник спрайта. Ведь если у нас будет второй игрок – у него может быть другой спрайт и другие размеры.

В control() происходит управление. Эта функция потом вызывается в update, что делает её всегда активной.

Далее идёт та же функция обработки коллизий (столкновений) с картой.

Затем функция update, в которой вызываются эти функции. Именно функция update вызывается в функции int main() внутри цикла “пока открыто окно”.

|  |
| --- |
| Имеется строка sprite.setOrigin(w/2, h/2), которая находится в конструкторе класса в самый конец (если она где-то в другом месте). Так вот эта строка задает середину спрайту. То есть его основная точка теперь не левый верхний угол, а середина. И когда спрайт коснется земли – он будет приподнят над ней из-за этой середины, потому что уже рисуется от середины, а раньше рисование было сверху слева, значит, при рисовании нужно сместиться ниже и правее. В общем, появится пустое пространство и будет пропасть между спрайтом и землей. Здесь просто нужно убрать эту пропасть. В ф-ции update нужно изменить строку задания позиции спрайту на такую:  sprite.setPosition(x+w/2, y+h/2); //задается позиция спрайта равная его середине. |
| В классе игрока видно, что при каждом нажатии клавиши переменная состояние объекта (state) принимает различные значения. В функции контроль можно увидеть условие выполнения движения и анимации персонажа: если нажата клавиша "стрелка влево", то переменная CurrentFrame увеличивается. Число 0.005 задает скорость анимации и домножение ее на время привязывает переменную к SFML времени. Следующее условие: если переменная CurrentFrame становится больше 3, то она откидывается на 3 назад. Это сделано для того, чтобы не уйти дальше третьей анимации.   sprite.setTextureRect(IntRect(40 \* int(CurrentFrame), 32, w, h));  Данная строка - это приведение типов. 40 умножается на целое число данного кадра, т.е. переменная CurrentFrame постепенно увеличивается от 0 до 3. Пока число от 0 до 1, нужно рисовать первый тайл. Затем переменная, благодаря строке выше, увеличивается от 1 до 2 и, следовательно, нужно начинать рисовать с координаты 40 (второй тайлик). Далее переменная текущего кадра и опять же, благодаря строке выше, принимает значение в интервале от 2 до 3, в этот момент 40 умножается на 2 и, соответственно, нужно переходить на третий тайлик. После чего переменная текущего кадра принимает значение 0 и нужно опять начинать с первого тайла.  После чего нужно двигать сам спрайт. На рисунке 2.3 изображены тайлы спрайта.    Рис. 3.3  Аналогично организовано передвижение персонажа и на другие клавиши.  Функция update, в качестве параметра она принимает время SFML, это сделано для того, чтобы функция update бесконечно работала пока открыто окно SFML. Так же благодаря этому времени данный объект оживляется и эта функция также способствует его оживлению. 3.2.3. Работа с камерой вида В заголовочном файле view.h объявляется объект, вид которого, по сути, и является камерой в SFML. Теперь в int main() создается размер вида камеры, т.е. инициализация камеры. Функция changeview позволят использовать разные возможности камеры.  if (Keyboard::isKeyPressed(Keyboard::U)) {  Масштабирование с уменьшением. Если нажата клавиша «U», то отдаляется камеру, далее записана скорость уменьшения.  if (Keyboard::isKeyPressed(Keyboard::R)) {  Поворот камеры. Если нажата клавиша «R», то поворачивается камеру.  if (Keyboard::isKeyPressed(Keyboard::L)) {  Поворот в обратную сторону.  if (Keyboard::isKeyPressed(Keyboard::I)) {  Размер камеры. Если нажата клавиша «I», то задается размер камеры.  if (Keyboard::isKeyPressed(Keyboard::P)) {  Другой размер камеры.  if (Keyboard::isKeyPressed(Keyboard::Q)) {  Разделение экрана.  Чтобы исключить возможность видеть края карты в самой первой функции, необходимо создать переменные, которые будут хранить временные координаты. Далее необходимо сделать условие, если координата по иксу меньше 640, то данная временная координата tempX принимает значение 640, после чего она передается в вид, т.е. меньше, чем 640 вид не передастся. Таким образом, нужно исключить тёмный экран слева. Аналогично по иксу справа и по игреку сверху и снизу. Если координата по иксу больше чем 640, то данная временная координата tempX принимает значение 640. Таким образом, нужно исключить првую тёмный экран справа. Далее необходимо убрать тёмный экран сверху и снизу. Если координата по игреку меньше, чем 360, то временная переменная tempY принимает значение 360. Тем самым исключается тёмный экран сверху. И если координата по игреку больше, чем 440, то временная переменная tempY принимает значение 440. Тем самым исключается тёмный экран снизу. 3.2.4. Взаимодействие с картой Поскольку данная карта массив, то её объекты это элементы массива.  В классе игрока в функции взаимодействия с картой. В этой функции нужно пройтись по всем элементам массива размером 32 на 32. Сначала нужно пройтись по игреку, а потом по иксу. Когда икс делится на 32, то получается левый квадратик, с которым персонаж соприкасается j < (x + w) / 32, - это самый правый квадратик, т.е. нужно идти от самого левого к самому правому. Внутри этого цикла, когда встречается один из символов: «0», «e», «a» или «b», то нужно проверить направление персонажа. dy и dx - это скорости в направлении, они принимают значение «speed». Если dy больше нуля, т.е. если нужно было идти вниз, выполняется это условие, то координата персонажа становится равна элементу массива минус высота персонажа, тем самым можно просто упереться в квадрат и не будет возможности идти дальше. Аналогично другим случаям.  Если встречается символ «m», то к переменной playerScore (она объявлена и инициализирована в классе игрока) прибавляется единица, тем самым увеличивается счетчик, т.е. если до того как нужно было собрать монету переменная равнялась 0, то теперь она будет равняться 1. После чего символ монеты заменяется на символ пробела, т.е. пропадает.  Если встречается символ «d», то убивается персонаж.  Не важно, с какой координаты начать проходить по элементам массива (x или y). Важно пройтись по обеим. На рисунке 2.4 изображена схема взаимодействия персонажа с картой.    Рис. 3.4  Далее, чтобы это функция заработала, её необходимо вызвать в функции update. 3.2.5. Работа с текстом Текст встречается во многих играх.  В функции bool startGame создается шрифт. Нужно загрузить шрифт, задать текст (по умолчанию он пустой) и размер текста. Можно задать ему красный текст, и сделать его жирным.  После чего объявляются переменные, формируется строка, задается текст и прибавляется переменная, т.е. конкатенируются строки. Задается позиция текста относительно центра камеры, затем всё это рисуется. 3.3 Выводы по главе 3 Разработанное приложение имеет множество функций, которые дополняют друг друга. Отличительной чертой является использование класса Entity. «Монетка» имеет простой интерфейс и проста своим геймплеем. В этой главе были пояснены все действия, которые происходят, непосредственно, в самом коде проекта. Так же проект можно неоднократно совершенствовать и вносить дополнительные функции. Можно так же сделать игровое поле размером больше предоставленного. В конечном, игру можно адаптировать под популярное приложение «Марио». |

# ЗАКЛЮЧЕНИЕ

Ни для кого не секрет, что игры уже давно укрепили свою позицию в современном обществе. Не существует, пожалуй, ни одного подростка, не игравшего когда-либо в игры. Кто-то находит в играх способ расслабиться, кто-то способ отвлечься от проблем, убить время. Но для некоторых игровые приложения стали целой жизнью, как для некоторых игроков, так и для программистов или дизайнеров, занимающихся разработкой игр и видящих себя в этом более чем на 10 лет вперёд.

Рынок игровых приложений с каждым годом становится всё шире, объёмнее. Игровая индустрия с каждым днём совершенствуется, что можно видеть просто посмотрев тот или иной трейлер к новинкам игр. Графика постепенно становится приближённой к реальности, а геймплей всё больше заставляет окунуться в игру с головой. Популярность игр и масштабность игрового сообщества обусловлена большой вариативностью и разнообразностью типов игр. Некоторые находят стоящим шутеры, кто-то приключения, ну а для кого-то 2D платформеры стали на первое место.

Платформеры появились в начале 1980-х, когда [игровые консоли](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BA%D0%BE%D0%BD%D1%81%D0%BE%D0%BB%D1%8C) не были достаточно мощными, чтобы отображать [трёхмерную](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) графику или видео. Они были ограничены статическими игровыми мирами, которые помещались на один экран, а [игровой герой](https://ru.wikipedia.org/w/index.php?title=%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B3%D0%B5%D1%80%D0%BE%D0%B9&action=edit&redlink=1) был виден в профиль. Персонаж лазал вверх и вниз по лестницам или прыгал с платформы на платформу, часто сражаясь с противниками и собирая предметы, улучшающие характеристики (так называемые «пауэрапы»). Первыми играми этого типа были [Space Panic](https://ru.wikipedia.org/wiki/Space_Panic) и [Apple Panic](https://ru.wikipedia.org/w/index.php?title=Apple_Panic&action=edit&redlink=1). За ними последовала игра [Donkey Kong](https://ru.wikipedia.org/wiki/Donkey_Kong), [аркадная игра](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%BA%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0) созданная фирмой «[Nintendo](https://ru.wikipedia.org/wiki/Nintendo)» и выпущенная в 1981 году. Вскоре процесс прохождения уровня перестал быть в основном вертикальным и стал горизонтальным с появлением длинных многоэкранных прокручивающихся игровых миров. Считается, что начало этому положила выпущенная фирмой [Activision](https://ru.wikipedia.org/wiki/Activision) в 1982 году игра [Pitfall!](https://ru.wikipedia.org/wiki/Pitfall!) для консолей [Atari 2600](https://ru.wikipedia.org/wiki/Atari_2600). [Manic Miner](https://ru.wikipedia.org/wiki/Manic_Miner) (1983) и её продолжение [Jet Set Willy](https://ru.wikipedia.org/wiki/Jet_Set_Willy) (1984) были наиболее популярными платформерами на [домашних компьютерах](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80_(%D0%B1%D1%8B%D1%82%D0%BE%D0%B2%D0%BE%D0%B9)).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Simple and Fast Multimedia Library [Electronic resource] / SFML Mode of access:

https://www.sfml-dev.org/index.php. Date of access: 26.05.2019

2. Колдаев, В.Д. Основы алгоритмизации и программирования / В.Д. Колдаев. Форум, Инфра-М, – 2006. – 414 с.

3. Пакина, Е.П. Основы алгоритмизации и программирования. Практикум Учебное пособие, – 2014. – 144 с.

4. Лафоре, Р.У. Объектно-ориентированное программирование в С++, – 2019. – 902 с.

5. Павловская, Т.А. C/C++. Процедурное и объектно-ориентированное программирование. Учебник для вузов, – 2015. – 496 с.

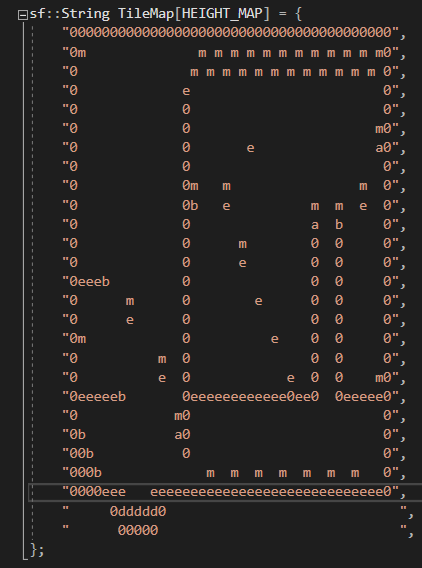
# ПРИЛОЖЕНИЕ 1

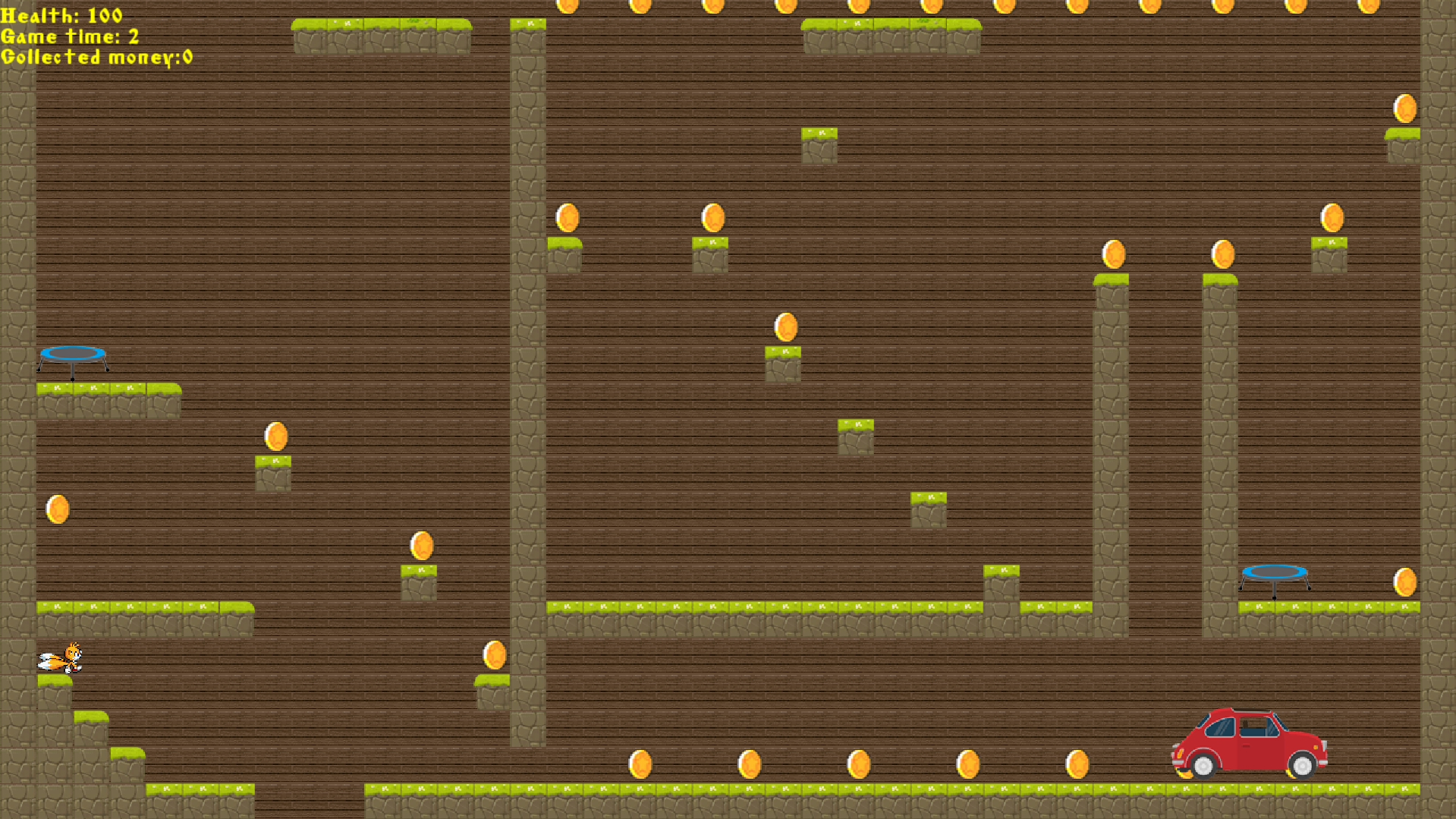
Меню игры.



# ПРИЛОЖЕНИЕ 2

**Игровое поле**





# ПРИЛОЖЕНИЕ 3

**Основные функции и логика игры.**

class Entity {

public:

float dx, dy, x, y, speed, moveTimer, CurrentFrame = 0;

int w, h, health;

bool life, isMove, onGround;

Texture texture;

Sprite sprite;

String name;// различим врагов по имени и дадим каждому свое действие в update в зависимости от имени

Entity(Image &image, float X, float Y, int W, int H, String Name)

{

x = X; y = Y; w = W; h = H; name = Name; moveTimer = 0;

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 getRect() {

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

}

};

////////////////////////////////////////////////////КЛАСС ИГРОКА////////////////////////

class Player :public Entity {

public:

enum { left, right, up, down, jump, stay } state;//добавляем тип перечисления - состояние объекта

int playerScore;//эта переменная может быть только у игрока

Player(Image &image, float X, float Y, int W, int H, String Name) :Entity(image, X, Y, W, H, Name) {

playerScore = 0;

state = stay;

if (name == "Player1") {

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

}

}

void control(float time) {

if (life) {

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

state = left; speed = 0.1;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(40 \* int(CurrentFrame), 32, w, h));

sprite.move(-0.1\*time, 0);

}

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

state = right; speed = 0.1;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(40 \* int(CurrentFrame), 0, w, h));

sprite.move(-0.1\*time, 0);

}

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

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

}

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

state = down;

}

changeview();

}

}

void update(float time)

{

control(time);//функция управления персонажем

switch (state)//тут делаются различные действия в зависимости от состояния

{

case right:dx = speed; break;//состояние идти вправо

case left:dx = -speed; break;//состояние идти влево

case up: break;//будет состояние поднятия наверх (например, по лестнице)

case down: dx = 0; break;//будет состояние во время спуска персонажа (например, по лестнице)

case stay: break;//и здесь то же

}

x += dx \* time;

checkCollisionWithMap(dx, 0);//обрабатываем столкновение по Х

y += dy \* time;

checkCollisionWithMap(0, dy);

sprite.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта в место его центра

if (health <= 0) {

life = false;

speed = 0;

}

if (!isMove) { speed = 0; }

//if (!onGround) { dy = dy + 0.0015\*time; }//убираем и будем всегда притягивать к земле

if (life) { getplayercoordinateforview(x, y); }

dy = dy + 0.0015\*time;//постоянно притягиваемся к земле

}

void checkCollisionWithMap(float Dx, float Dy)//ф-ция проверки столкновений с картой

{

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

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

{

if (TileMap[i][j] == '0' || TileMap[i][j] == 'e' || TileMap[i][j] == 'a' || TileMap[i][j] == 'b')

{

if (Dy > 0) { y = i \* 32 - h; dy = 0; onGround = true; }//по Y вниз=>идем в пол (стоим на месте) или падаем. В этот момент надо вытолкнуть персонажа и поставить его на землю, при этом говорим, что мы на земле, тем самым снова можем прыгать

if (Dy < 0) { y = i \* 32 + 32; dy = 0; }//столкновение с верхними краями карты (может и не пригодиться)

if (Dx > 0) { x = j \* 32 - w; }//с правым краем карты

if (Dx < 0) { x = j \* 32 + 32; }// с левым краем карты

}

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

{

playerScore++;

TileMap[i][j] = ' ';

}

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

{

health = -100;

//TileMap[i][j] = ' ';

}

}

}

};

class Enemy :public Entity {

public:

Enemy(Image &image, float X, float Y, int W, int H, String Name) :Entity(image, X, Y, W, H, Name) {

if (name == "EasyEnemy") {

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

dx = 0.1;//даем скорость. этот объект всегда двигается

}

if (name == "jumpImage")

{

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

}

}

void checkCollisionWithMap(float Dx, float Dy)//ф-ция проверки столкновений с картой

{

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

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

{

if (TileMap[i][j] == '0' || TileMap[i][j] == 'e' || TileMap[i][j] == 'a' || TileMap[i][j] == 'b')//если элемент наш тайлик земли, то

{

if (Dy > 0) { y = i \* 32 - h; }//по Y вниз=>идем в пол (стоим на месте) или падаем. В этот момент надо вытолкнуть персонажа и поставить его на землю, при этом говорим, что мы на земле, тем самым снова можем прыгать

if (Dy < 0) { y = i \* 32 + 32; }//столкновение с верхними краями карты (может и не пригодиться)

if (Dx > 0) { x = j \* 32 - w; dx = -0.1; sprite.scale(-1, 1); }//с правым краем карты

if (Dx < 0) { x = j \* 32 + 32; dx = 0.1; sprite.scale(-1, 1); }// с левым краем карты

}

}

}

void update(float time)

{

if (name == "EasyEnemy") {

moveTimer += time;if (moveTimer>3000){ dx \*= -1; moveTimer = 0; sprite.scale(-1, 1);}//меняет направление примерно каждые 3 сек

checkCollisionWithMap(dx, 0);//обрабатываем столкновение по Х

x += dx \* time;

sprite.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта в место его центра

}

}

void updateJump(float time)

{

if (name == "jumpImage") {//для персонажа с таким именем логика будет такой

sprite.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта в место его центра

}

}

};

class MovingPlatform : public Entity {//класс движущейся платформы

public:

MovingPlatform(Image &image, float X, float Y, int W, int H, String Name) :Entity(image, X, Y, W, H, Name) {

sprite.setTextureRect(IntRect(0, 0, W, H));//прямоугольник

dx = 0.08;//изначальное ускорение по Х

}

void checkCollisionWithMap(float Dx, float Dy)//ф-ция проверки столкновений с картой

{

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

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

{

if (TileMap[i][j] == '0' || TileMap[i][j] == 'e' || TileMap[i][j] == 'a' || TileMap[i][j] == 'b')//если элемент наш тайлик земли, то

{

if (Dy > 0) { y = i \* 32 - h; }//по Y вниз=>идем в пол (стоим на месте) или падаем. В этот момент надо вытолкнуть персонажа и поставить его на землю, при этом говорим, что мы на земле, тем самым снова можем прыгать

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

if (Dx > 0) { x = j \* 32 - w; dx = -0.1; sprite.scale(-1, 1); }

if (Dx < 0) { x = j \* 32 + 32; dx = 0.1; sprite.scale(-1, 1); }

}

}

}

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

{

x += dx \* time;

checkCollisionWithMap(dx, 0);

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

}

void updatePlatform(float time)

{

if (name == "MovingPlatform") {//для персонажа с таким именем логика будет такой

sprite.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта в место его центра

}

}

};

bool startGame()

{

RenderWindow window(VideoMode(1920, 1080), "My game", Style::Fullscreen);

view.reset(FloatRect(0, 0, 1280, 720));

menu(window);

Image map\_image;

map\_image.loadFromFile("map\_.png");

Texture map;

map.loadFromImage(map\_image);

Sprite s\_map;

s\_map.setTexture(map);

Image heroImage;

heroImage.loadFromFile("MilesTailsPrower.png");

Player p(heroImage, 32, 700, 40, 28, "Player1");

Image easyEnemyImage;

easyEnemyImage.loadFromFile("shamaich.png");

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

Enemy easyEnemy(easyEnemyImage, 750, 701, 138, 64, "EasyEnemy");

Image movePlatformImage;

movePlatformImage.loadFromFile("platform.png");

MovingPlatform movePlatform(movePlatformImage, 32, 96, 160, 32, "MovingPlatform");

Image movePlatformImage2;

movePlatformImage2.loadFromFile("platform.png");

MovingPlatform movePlatform2(movePlatformImage2, 480, 96, 160, 32, "MovingPlatform");

Image jumpImageImage;

jumpImageImage.loadFromFile("jump.png");

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

Enemy jumpImage(jumpImageImage, 32, 384, 64, 32, "jumpImage");

Image jumpImageImage2;

jumpImageImage2.loadFromFile("jump.png");

jumpImageImage2.createMaskFromColor(Color(255, 0, 0));

Enemy jumpImage2(jumpImageImage2, 1088, 576, 64, 32, "jumpImage");

Font font, font2;

font.loadFromFile("CyrilicOld.ttf"); font2.loadFromFile("CyrilicOld.ttf");

Text text("", font, 20); Text text2("", font2, 60);

text.setFillColor(Color::Yellow); text2.setFillColor(Color::Red);

text.setStyle(Text::Bold | Text::Underlined);text2.setStyle(Text::Bold | Text::Underlined);

Clock clock;

Clock gameTimeClock;

int gameTime = 0;

while (window.isOpen())

{

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

if (p.life) gameTime = gameTimeClock.getElapsedTime().asSeconds();//игровое время в секундах идёт вперед, пока жив игрок.

clock.restart();

time = time / 800;

Event event;

while (window.pollEvent(event))

{

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

window.close();

}

if ((movePlatform.name == "MovingPlatform") && (movePlatform.getRect().intersects(p.getRect())))//если игрок столкнулся с объектом списка и имя этого объекта movingplatform

{

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

if (p.y + p.h < movePlatform.y + movePlatform.h)//если игрок находится выше платформы

{

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

}

}

if ((movePlatform2.name == "MovingPlatform") && (movePlatform2.getRect().intersects(p.getRect())))//если игрок столкнулся с объектом списка и имя этого объекта movingplatform

{

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

if (p.y + p.h < movePlatform2.y + movePlatform2.h)//если игрок находится выше платформы

{

p.y = movePlatform2.y - p.h + 3; p.x += movePlatform2.dx\*time; p.dy = 0; p.onGround = true; // то выталкиваем игрока так, чтобы он как бы стоял на платформе

}

}

if (easyEnemy.getRect().intersects(p.getRect()))//если прямоугольник спрайта объекта пересекается с игроком

{

if (easyEnemy.name == "EasyEnemy") {//и при этом имя объекта EasyEnemy,то..

if (p.dx <= 0) { p.x = easyEnemy.x + easyEnemy.w; }//если столкнулись с врагом и игрок идет влево, то выталкиваем игрока

if (p.dx >= 0) { p.x = easyEnemy.x - p.w; }//если столкнулись с врагом и игрок идет вправо, то выталкиваем игрока

}

}

if (jumpImage.getRect().intersects(p.getRect()) || jumpImage2.getRect().intersects(p.getRect()))//если прямоугольник спрайта объекта пересекается с игроком

{

if (jumpImage.name == "jumpImage" || jumpImage2.name == "jumpImage") {//и при этом имя объекта jumpImage,то..

if ((p.dy > 0) && (p.onGround == false)) {//если прыгнули на батут, то даем врагу скорость 0,отпрыгиваем от него чуть вверх, даем ему здоровье 0

p.dy = -1;

}

}

}

if (Keyboard::isKeyPressed(Keyboard::Tab)) { return true; }

if (Keyboard::isKeyPressed(Keyboard::Escape)) { return false; }

p.update(time);

jumpImage.updateJump(time);

jumpImage2.updateJump(time);

movePlatform2.update(time);

movePlatform.update(time);

easyEnemy.update(time);

window.setView(view);

window.clear();

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

{

if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(0, 0, 32, 32));

if (TileMap[i][j] == 'm') s\_map.setTextureRect(IntRect(32, 0, 32, 32));

if ((TileMap[i][j] == 'f')) s\_map.setTextureRect(IntRect(96, 0, 32, 32));

if ((TileMap[i][j] == 'h')) s\_map.setTextureRect(IntRect(128, 0, 32, 32));

if ((TileMap[i][j] == 'a')) s\_map.setTextureRect(IntRect(192, 0, 32, 32));

if ((TileMap[i][j] == 'e')) s\_map.setTextureRect(IntRect(224, 0, 32, 32));

if ((TileMap[i][j] == 'b')) s\_map.setTextureRect(IntRect(320, 0, 32, 32));

if ((TileMap[i][j] == '0')) s\_map.setTextureRect(IntRect(352, 0, 32, 32));

if ((TileMap[i][j] == 'd')) s\_map.setTextureRect(IntRect(448, 0, 32, 32));

s\_map.setPosition(j \* 32, i \* 32);

window.draw(s\_map);

}

ostringstream playerHealthString, gameTimeString, playerScoreString;

playerHealthString << p.health; gameTimeString << gameTime; playerScoreString << p.playerScore;

text.setString("Health: " + playerHealthString.str() + "\nGame time: " + gameTimeString.str() + "\nCollected money:" + playerScoreString.str());

text.setPosition(view.getCenter().x - 640, view.getCenter().y - 360);

if (p.life <= 0) {

text2.setString("YOU DEAD\n Enter the TAB");

text2.setPosition(view.getCenter().x - 200, view.getCenter().y - 100);

}

if (p.playerScore == 44) {

return false;

}

window.draw(text);

window.draw(text2);

window.draw(movePlatform.sprite);

window.draw(movePlatform2.sprite);

window.draw(jumpImage.sprite);

window.draw(jumpImage2.sprite);

window.draw(easyEnemy.sprite);

window.draw(p.sprite);

window.display();

}

}

void gameRunning() {//ф-ция перезагружает игру , если это необходимо

if (startGame()) { gameRunning(); }////если startGame() == true, то вызываем занова ф-цию isGameRunning, которая, в свою очередь, опять вызывает startGame()

}

int main()

{

gameRunning();

return 0;

}

menu.h (меню игры)

void menu(RenderWindow & window) {

Texture menuTexture1, menuTexture2, aboutTexture, menuBackground;

menuTexture1.loadFromFile("NewGame.png");

menuTexture2.loadFromFile("Exit.png");

menuBackground.loadFromFile("Penguins.jpg");

Sprite menu1(menuTexture1), menu2(menuTexture2), menuBg(menuBackground);

bool isMenu = 1;

int menuNum = 0;

menu1.setPosition(100, 30);

menu2.setPosition(100, 150);

menuBg.setPosition(345, 0);

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

while (isMenu)

{

menu1.setColor(Color::White);

menu2.setColor(Color::White);

menuNum = 0;

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

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

if (IntRect(100, 150, 300, 50).contains(Mouse::getPosition(window))) { menu2.setColor(Color::Blue); menuNum = 2; }

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

{

if (menuNum == 1) isMenu = false;

if (menuNum == 2) { window.close(); isMenu = false; }

}

window.draw(menuBg);

window.draw(menu1);

window.draw(menu2);

window.display();

}

}

view.h (камера вида)

View view;

void getplayercoordinateforview(float x, float y) { //функция для считывания координат игрока

float tempX = x; float tempY = y;//считываем коорд игрока и проверяем их, чтобы убрать края

if (x < 640) tempX = 640;//убираем из вида левую сторону

if (x > 640) tempX = 640;

if (y < 360) tempY = 360;

if (y > 440) tempY = 440;

view.setCenter(tempX, tempY); //следим за игроком, передавая его координаты.

}

void changeview() {

if (Keyboard::isKeyPressed(Keyboard::U)) {

view.zoom(1.0020f); //масштабируем, уменьшение

//view.zoom(1.0006f); //то же самое помедленнее соответственно

}

if (Keyboard::isKeyPressed(Keyboard::R)) {

//view.setRotation(90);//сразу же задает поворот камере

view.rotate(-1);//постепенно поворачивает камеру (отрицательное значение - в обратную сторону)

}

if (Keyboard::isKeyPressed(Keyboard::L)) {

//view.setRotation(90);//сразу же задает поворот камере

view.rotate(1);//постепенно поворачивает камеру (отрицательное значение - в обратную сторону)

}

if (Keyboard::isKeyPressed(Keyboard::I)) {

view.setSize(1280, 720);//устанавливает размер камеры (наш исходный)

}

if (Keyboard::isKeyPressed(Keyboard::P)) {

view.setSize(540, 380);//например, другой размер

}

if (Keyboard::isKeyPressed(Keyboard::Q)) {

view.setViewport(sf::FloatRect(0, 0, 0.5f, 1));//таким образом, делается раздельный экран для игры на двоих. нужно только создать ещё один объект View и привязывать к нему координаты игрока 2.

}

}

map.h (карта)

const int HEIGHT\_MAP = 27;//размер карты высота

const int WIDTH\_MAP = 40;//размер карты ширина

sf::String TileMap[HEIGHT\_MAP] = {

"0000000000000000000000000000000000000000",

"0m m m m m m m m m m m m m0",

"0 m m m m m m m m m m m m 0",

"0 e 0",

"0 0 0",

"0 0 m0",

"0 0 e a0",

"0 0 0",

"0 0m m m 0",

"0 0b e m m e 0",

"0 0 a b 0",

"0 0 m 0 0 0",

"0 0 e 0 0 0",

"0eeeb 0 0 0 0",

"0 m 0 e 0 0 0",

"0 e 0 0 0 0",

"0m 0 e 0 0 0",

"0 m 0 0 0 0",

"0 e 0 e 0 0 m0",

"0eeeeeb 0eeeeeeeeeeee0ee0 0eeeee0",

"0 m0 0",

"0b a0 0",

"00b 0 0",

"000b m m m m m m m 0",

"0000eee eeeeeeeeeeeeeeeeeeeeeeeeeeeee0",

" 0ddddd0 ",

" 00000 ",

};