Министерство образования Московской области

**Государственное автономное профессиональное образовательное**

**учреждение Московской области «Егорьевский техникум»**

Допущена к защите

заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_ С.Г. Большова

«\_\_\_» июня 2023г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(ДИПЛОМНАЯ РАБОТА)**

**на тему: «Разработка игрового движка Noka для разработки игры платформеры»**

**Студент** Величко Дмитрий Олегович

**Специальность** 09.02.07 Информационные системы и программирование

**Группа** Ип-93

**Руководитель** В.А. Тараканова

г. Егорьевск, 2023 г.

**Содержание**

[**Введение** 3](#_Toc131085759)

[**Глава 1. Игровой движок** 4](#_Toc131085760)

[**1.1. Особенности популярных игровых движков Unity и Unreal Engine 4** 5](#_Toc131085761)

[**1.2. Как создаются игры с помощью игровых движков** 6](#_Toc131085762)

[**Глава 2. Окна и графика** 8](#_Toc131085763)

[**2.1. Графические библиотеки для языка программирования C++** 9](#_Toc131085764)

[**2.2. Подключение SFML** 11](#_Toc131085765)

[**2.3. Работа с SFML** 12](#_Toc131085766)

[**Глава 3. Игровой персонаж** 16](#_Toc131085767)

[**3.1. Внешний вид и управление** 17](#_Toc131085768)

[**Глава 4. Игровая локация** 25](#_Toc131085769)

[**4.1. Локация в играх жанра плафтормеры.** 26](#_Toc131085770)

[**4.2. Tiled. Тайловые карты** 28](#_Toc131085771)

[**4.2. RapidXml. Парсинг xml файла** 31](#_Toc131085772)

[**Глава 5. Физика** 35](#_Toc131085773)

[**Глава 6. Дополнительные механики** 36](#_Toc131085774)

[**Заключение** 37](#_Toc131085775)

[**Список литературы** 38](#_Toc131085776)

[**Листинг** 39](#_Toc131085777)

[**Экономическая часть** 40](#_Toc131085778)

# **Введение**

Актуальность дипломного проекта обусловлена малым количеством русскоязычных работ на тему разработки игр на базовом уровне, а, то есть на уровне написания кода, без использования готовых движков, чаще всего полностью скрывающих реализацию проекта за своими абстракциями, позволяющими изменять игровую сцену просто перетягивая объекты мышкой.

Цель дипломного проекта заключается в разработке собственного игрового движка на языке программирования C++ с использованием OpenSours фреймворков и библиотек, а так же изучение работы масштабных игровых движков на низком уровне реализации.

В соответствии с целью поставлены следующие задачи:

* изучение работы с графикой и окном приложения через код;
* изучение понятия дизайн уровней и разработка автоматизации построения игровой карты;
* разработка персонажа и управления им;
* изучение способа построения игровой физики в 2d пространствах и ее реализация.

Объектом исследования диплома можно считать разработку игр, а предметом ее исследования - игровые движки с двумерной графикой.

Интерес в написании данной дипломной работы на данную тему заключен в:

- повышении опыта программирования на языке C++;

- изучении работы игровых движков на уровне кода;

- изучении новых фреймворков и библиотек;

# **Глава 1. Игровой движок**

Термин «игровой движок» является прямой копией английского «Game Engine». Фактически это объединенный в единое целое комплекс прикладных программ, с помощью которых обеспечивается графическая визуализация, звуковое сопровождение, перемещение внутриигровых персонажей, их действия в соответствии со скриптами, а также игра по сети, встроенные графические сцены, соблюдение физических эффектов и законов и многое другое.

Впервые этот термин появился в середине 90-х годов прошлого века. Связан он был с играми в жанре «шутер от первого лица», а точнее, с самым популярным на тот момент Doom. Исходный код этой игры был построен очень продуманно, с выделенными основными компонентами: системой трехмерной графики, звуками, расчетами столкновений, скриптами. Это привело к тому, что игровые программисты не стали писать свой код, а использовали наработки разработчиков Doom, внеся в код некоторые изменения: рисовали другие уровни, изменяли графику и внешний вид оружия, корректировали правила и выпускали новые игры, в основе которых оставался все тот же код из Doom.

Есть ли польза от использования готового игрового движка? Несомненно, разработчик получает готовый качественный инструмент с большим количеством библиотек. В результате ему не надо писать большую часть базового программного кода и можно сосредоточиться на реализации своих идей, графики, игровой механики и сюжета, не тратя время на написание кода с нуля.

В результате ряд компаний занялся разработкой именно игровых движков, а разработчики игр стали покупать на них лицензии, как это получилось с Unreal Engine или id Tech 3. Стоимость лицензии может составлять от нескольких тысяч до миллионов долларов. Но при этом надо отметить, что для некоммерческого использования многие игровые движки, например, популярные Unity и Unreal Engine 4 доступны бесплатно. Остановимся на этих движках немного подробнее.

## **1.1. Особенности популярных игровых движков Unity и Unreal Engine 4**

**Unreal Engine 4**

Этот движок смело можно назвать легендой. Его разработка началась в 1998 году и с тех пор он постоянно модернизируется, дополняется и совершенствуется. Современный Unreal Engine 4 — это движок, на котором пишут игры для любых платформ и операционных систем, начиная от ОС Windows и заканчивая всеми современными консолями, а также мобильными платформами, в том числе и iOS.

13 мая 2020 года разработчики из Epic Games анонсировали Unreal Engine 5 и показали демо для консоли Playstation 5 и на данный момент времени уже закончили его разработку. В настоящее время он бесплатен при условии, что выпущенные приложения приносят прибыль не более $3 000 в квартал. Если сумма прибыли больше, потребуется лицензировать движок и выплачивать его разработчикам процент от прибыли.

**Unity**

Unity — одна из популярных платформ для разработчиков игровых приложений. Можно услышать, что этот движок называют самым молодым. Но тут надо отметить, что он появился в 2005 году и с тех пор успешно развивается.

Как и Unreal Engine, Unity — кроссплатформенная система. На ней пишут игры для любых платформ и в любых жанрах. На Unity делают как простые аркады и головоломки, так и шутеры от первого лица с достаточно сложным игровым миром. Пример — Dead Trigger.

Большой плюс Unity — простота его освоения. Минус — графика в играх, созданных на основе этого движка. Она выглядит проще и не настолько реалистична, как у Unreal Engine. Тем не менее, около половины всех мобильных игр, по заверениям разработчиков, написаны именно на этом движке.

## **1.2. Как создаются игры с помощью игровых движков**

Для разработчика игровых приложений движки представляют собой программную среду, в которой он ведет разработку проекта. Ее использование позволяет не заниматься такими рутинными вещами, как описание работы с графикой, звуком и физической моделью. Но это не значит, что программировать не придется ничего. Разработчику все равно потребуется писать скрипты для внутриигровых действий. На Unity, например, потребуется работа с C#, да и на Unreal Engine знание языков программирования не помешает.

Необходимо отметить, что важной особенностью Unreal Engine является технология Blueprints, позволяющая описывать игровую логику и события с помощью графических схем, без использования языков программирования. Это, конечно, приведет к тому, что созданная игра будет занимать больше места и требовать более быстрой платформы, но зато процесс разработки значительно упрощается.

Использование игровых движков позволяет избавиться от написания кода для очень многих рутинных моментов, так как, кроме самих движков, для них существует огромное количество библиотек и расширений. С их помощью первые простейшие игры на Unity можно создать уже через несколько часов изучения платформы. Специально для начинающих в Unity существует масса проектов вроде Creator Kit и Microgame, предлагающих большое количество исходных материалов для написания простых приложений в 2D и 3D. На Unreal Engine также есть множество библиотек и уроков, позволяющих быстро освоить программную среду и начать писать простые игровые приложения.

Все игровые движки позволяют добавлять и рисовать уровни, их элементы и персонажей, прописывать события, которые происходят в зависимости от действий главного героя. При этом разработчик не тратит время на написание сотен строчек кода, а занимается только реализацией своих непосредственных идей.

Получается, что игровой движок с точки зрения разработчика является программной платформой, на которой ведется разработка приложения. Кстати, это совсем не обязательно игра. Unity, например, активно используют в работе над приложениями с дополненной реальностью. А это уже не только игры, но и путеводители, справочники, энциклопедии и многое другое.

А с точки зрения игрока или пользователя написанного приложения, игровой движок — это основа игры, на которую разработчиком наложен сюжет, уровни, графика и музыка. Разница между этими двумя определениями небольшая, но она все-таки есть.

# **Глава 2. Окна и графика**

Графика – один из самых важных элементов в компьютерных играх. Из игры в игру, из проекта в проект, в зависимости от концепта и задумки важность этого элемента может разниться, но в независимости от этого в современных играх без графики никуда, ведь она предоставляет целый пласт возможностей, иногда даже более объемный чем текст, геймплей и звук.

И так, прежде чем начать перечисление популярных графических библиотек, стоит упомянуть то, из чего они все, по большей части состоят.

Графические API: OpenGL и DirectX являются наиболее популярными способами обращения компьютера к графическому процессору видеокарты. Представляют они собой список возможных обращений к GPU, написанный разработчиками видеокарты и приведенный к стандартизированному виду. Именно из-за приведения списка обращений к единому стандарту, появляется такая вещь как кросплатформенность – возможность запуска программного обеспечения на множестве платформ. Естественно работа с графическими API довольно трудоемка и имеет множество тонкостей, но изучение и совершенствование навыков в этом направлении позволит программисту использовать мощности графического процессора так как ему заблагорассудиться.

Для того чтобы обращение к GPU было выполнено, оно должно получить контекст своей работы, которым в случае c OC Windows будет являться его стандартное окно. Для создания такого окна существует несколько способов, начиная с использования непосредственно WinAPI, библиотеки функций и обращений к Windows, и заканчивая специализированными библиотеками, упрощающими большую часть взаимодействия с API операционной системы, например, glfw.

Теперь, немного лучше понимая работу низкоуровнего графического API, а также то, что для него необходим контекст в виде окна приложения, можно упростить написание движка, использовав для этого полноценный фреймворк.

## **2.1. Графические библиотеки для языка программирования C++**

Для языка программирования C++ существует множество фреймворков с открытым исходным кодом направленных как на поддержку открытия окон приложений и вывода в них двумерной и трехмерной графики, так и на комплекс функций способствующих упрощению разработки игр на уровне кода. Ниже будут рассмотрены несколько из популярных фреймворков.

**wxWidgets**

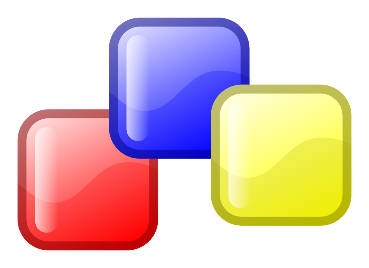


Рисунок 1.Подписать

[wxWidgets](https://www.wxwidgets.org/) — одна из старейших, но в то же время наиболее известных графических библиотек для отрисовки GUI. Её основной плюс — производительность.

Важная особенность wxWidgets заключается в использовании «родных» графических элементов интерфейса операционной системы везде, где это возможно. Это существенное преимущество для многих пользователей, поскольку они привыкают работать в конкретной среде, а изменения интерфейса программ часто вызывают затруднения в их работе.

**SDL 2.0**



Рисунок 2

[SDL](https://www.libsdl.org/) — кроссплатформенная 2D-библиотека, предназначенная для написания приложений (в основном игр). Поддерживаются Windows, Linux, Mac OS, а также Android, Windows Mobile и iOS. Преимущества библиотеки — быстрота, надёжность и лёгкость в эксплуатации. Также в библиотеке реализованы профессиональные инструменты для работы со звуком — это большой плюс при разработке крупных проектов.Комбинируется с wxWidgets и хорошо интегрируется с OpenGL.

**SFML**



Рисунок 3

[SFML](https://www.sfml-dev.org/) (Simple and Fast Multimedia Library) — одна из самых удобных и быстрых графических библиотек для C++. Её неоспоримое преимущество — минимальные требования к уровню знаний языка и лёгкость освоения: всего за несколько дней можно написать вполне полноценную программу. К примеру, [первое графическое приложение](https://www.sfml-dev.org/tutorials/2.5/start-vc.php) «SFML works!» можно написать всего за минуту. Ещё одно преимущество — кроссплатформенность: SFML работает под Windows, Linux (только X11, но не Wayland) и Mac OS, [планируется](https://www.sfml-dev.org/index.php) выход под Android и iOS.

SFML популярен среди небольших команд благодаря тому, что разработка графической части программы не требует написания больших объёмов кода.

Именно из-за своей простоты и удобства SFML был выбран для этого проекта при сравнении с более сложным и глубоким SDL. Данный фреймворк поможет погрузиться в разработку без сильной потери времени на изучение, но для начала его стоит установить и подключить к проекту.

## **2.2. Подключение SFML**

Устанавливается фреймворк просто, по нажатию кнопки на официальном сайте [SFML](https://www.sfml-dev.org), во вкладке Download находятся ссылки на последнюю и прочие версии библиотеки. После установки программист получает папку в которой есть все что требуется для подключения. Библиотека уже собрана и не требует каких-либо манипуляций над ней.

Далее следует подключение. В среде разработки Visual Studio 2019 создается пустой консольный проект на языке C++. Сразу же после этого в нем создается первый рабочий файл с расширением .cpp, в котором следует прописать какой-либо базовый код, например “Hello, world!”. Требуется это для сборки проекта, ведь без нее не получится добраться до нужных настроек среды разработки.

Подключая библиотеку в первый раз, можно не понять что все-таки нужно сделать для этого, не получив в процессе несколько десятков ошибок. В сети довольно трудно найти гайд с последовательностью действий. Но в сети его быть и не должно, по той причине, что он расположен на том же самом сайте где и скачиваемый фреймворк.

Так вот, на все том же официальном сайте во вкладке Learn есть вся документация библиотеки, а также раздел Tutorial, в котором вполне доходчиво расписано как подключить фреймворк sfml к большинству сред разработки.

Помимо последовательности шагов подключения на странице обучения присутствует тестовый код, который поможет проверить правильность подключения. При выполнении программа создаст окно и выведет в нем зеленый круг.

## **2.3. Работа с SFML**

В фреймворке sfml существует как ручное рисование изображений в окне при помощи функций основанных на vertex-ах OpenGl. Vertex – точка в трехмерном пространстве на основе которой ведется построение ребер, плоскостей, а затем и фигур. Так и вывод уже готового изображения.

Вывод заранее нарисованного изображения в окно происходит посредствам помещения его Texture (текстуры), загруженного в память представления изображения в двоичном виде, в объект Sprite (спрайт). Данный объект представляет собой многоугольник Rect с установленным размером, положением в файле и загруженной в него текстурой.



Рисунок 4

Стоит отметить, что в фреймворке SFML, часто встречается ошибка выхода текстуры из области видимости главной функции. В таком случае все связанные с этой текстурой спрайты в окне будут выглядеть как белые квадраты или прямоугольники. Для того чтобы это не происходило нужно создать вспомогательный класс, в проекте названный TextureHolder и содержащий в себе контейнер Map, связанный список. Именно в этом контейнере будут храниться текстуры, не теряясь до завершения программы.

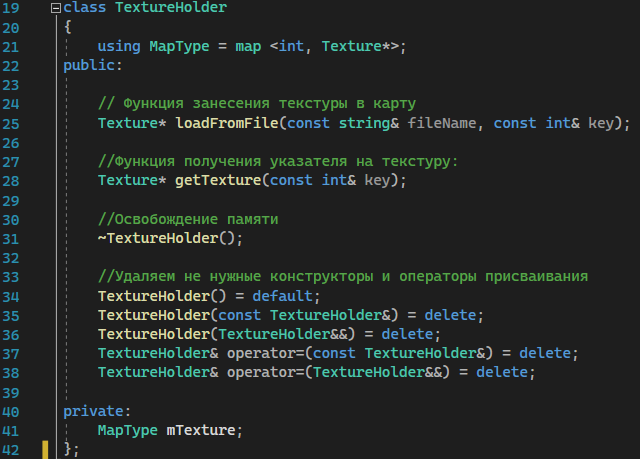


Рисунок 5

За ненадобностью из класса удаляются все ненужные операторы, а также конструктор. Далее, очередь деструктора, цель которого освобождение памяти и двух функций, создания или занесения текстуры в контейнер и получения указателя на нужную текстуру при помощи их цифровой индексации, главной особенности контейнера связанного списка. В данном случае нам нужен именно указатель на текстуру, из-за устройства конструктора класса Sprite.

Помимо контейнера текстур, были созданны четыре перегруженные функции, упрощающие и ускоряющие создание спрайтов в большем их количестве. Каждая функция принимает все больше данных уточняющих: pos\_OnWindow - положение в окне, size - размер изображения, pos\_OnFile - позицию изображения в файле

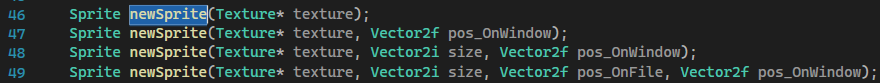


Рисунок 6

Типы данных этих переменных являются двумерными векторами - двумя числами определенного типа, указывающими на парные характеристики какого-либо объекта: Vector2f – вектор типа float, Vector2i – вектор типа int.

Например, вектор размера, size, представляет собой два числа, одно из которых длинна прямоугольника Rect, а другое его ширина.

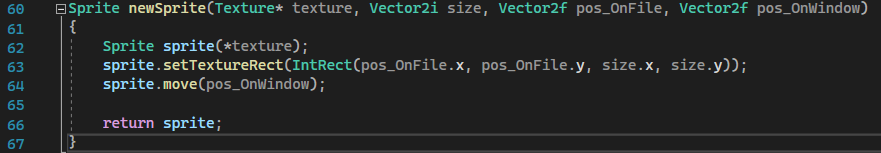


Рисунок 7

В данной функции создания спрайта с дополнительными условиями, показано как происходит работа с этим объектом, а именно установка Rect прямоугольника, его положения в файле и его размера в нем. После же происходит передвижение изображения в окне на определенное количество пикселей. Пиксели являются основной мерой длинны в SFML, помимо пикселей есть только градусы и радианы, дли измерения угла поворота. Стоит заметить, что функция move лишь передвигает объект из уже установленной точки в новую, путем сложения координат старого положения и введенных в эту функцию. Для точной установки позиции, следует использовать функцию setPosition() и лишь в случае когда позиция объекта равна нулевой [0,0] эти две функции будут равнозначными.

Далее речь пойдет о том, зачем вообще указывать положение изображения в его родном файле и на самом деле все просто. Это нужно для того, чтобы достать нужное изображение из большой картинки. Ведь не всегда файл изображения содержит только одно, нужное разработчику, а зачастую художник игровых текстур, специально помещает все изображения в один файл, на один сплошной лист. Такой файл называется Тайлсетом – Tileset, а одна отдельная текстура на нем является тайлом –Tile.

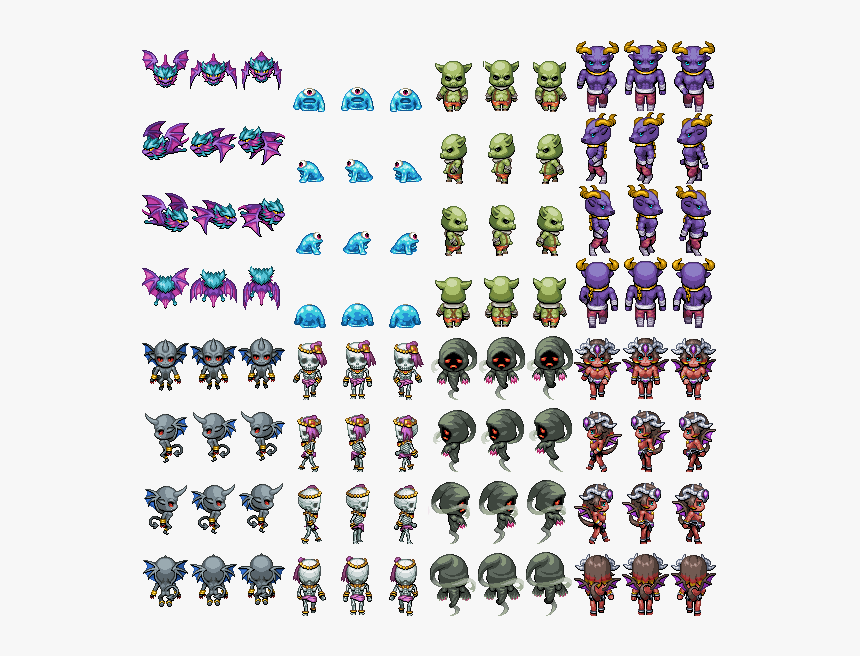


Рисунок 8

В тайлсеты можно поместить любое количество изображений, но лучше в данном деле руководствоваться удобством использования, а не пытаться уместить все текстуры на одном листе. Чаще всего текстуры помещают в тайлсет по принадлежности к определенному типу или объекту. Например, кадры анимаций персонажа или же тайлы карты первого игрового уровня, о которых будет идти речь в главе 4.

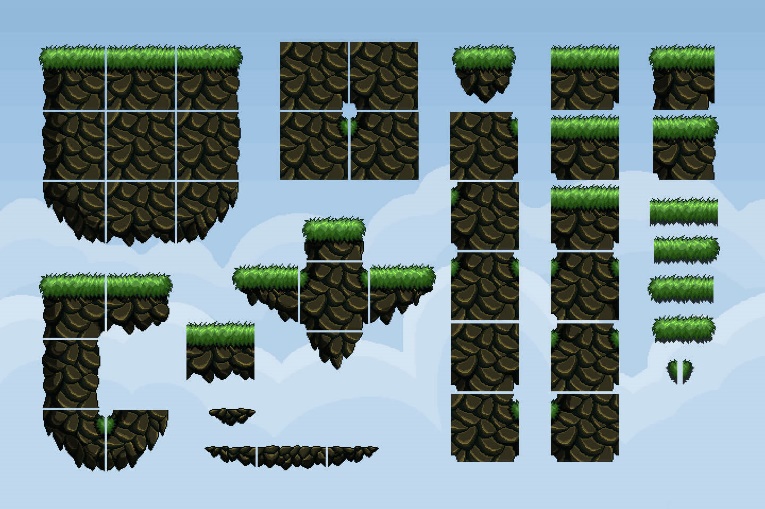
****

Рисунок 9

# **Глава 3. Игровой персонаж**

Игрово́й персона́ж — обозначение персонажа в компьютерных играх, который управляется человеком-игроком. Управление человеком отделяет игровых персонажей от неигровых, управляемых игровым искусственным интеллектом. В подавляющем большинстве случаев игровой персонаж является главным героем игры.

Игровой персонаж может быть, как полностью вымышленным, так и основываться на реальных личностях. В спортивных симуляторах, таких как симулятор бокса, футбола или хоккея, в большинстве случаев игровой персонаж имеет внешний вид реальных спортсменов. В играх с историческим сюжетом игровой персонаж может быть какой-то значимой исторической персоной. Например, в разных стратегиях игрок может управлять знаменитыми полководцами и государственными деятелями.

Исходя из того, чем является игровой персонаж, требуется создать для него: внешний вид, управление, характеристики, ограничения и способы взаимодействия с окружением.

## **3.1. Внешний вид и управление**

Главный герой игры внешне выглядит как рыцарь, с пером на шлеме.



Рисунок 10

Как и вся остальная пиксельная графика он был нарисован в специальном пиксельном редакторе Aseprite. Сначала была создана стандартная позиция, по которой далее были нарисованы все необходимые анимации. В их числе: спокойствие, бег, прыжок и падение.

Для того чтобы отделить все связанное с персонажем игрока, в коде игры прописываем класс Player. В нем будут содержаться все необходимые переменные, начиная с направления и заканчивая количеством собранных монеток.

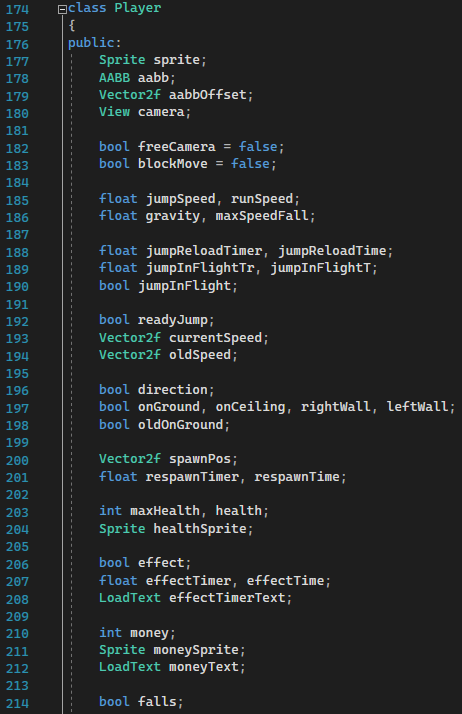


Рисунок 11

Так же, необходимо создать конструктор для этого класса, в котором будут инициализироваться все эти переменные. После создания конструктора начинается, создание методов, первые два из них будут отвечать за управление персонажем и обновление его игровых данных, характеристик.

В методе управления нет ничего сложного, кроме, возможно прыжка.

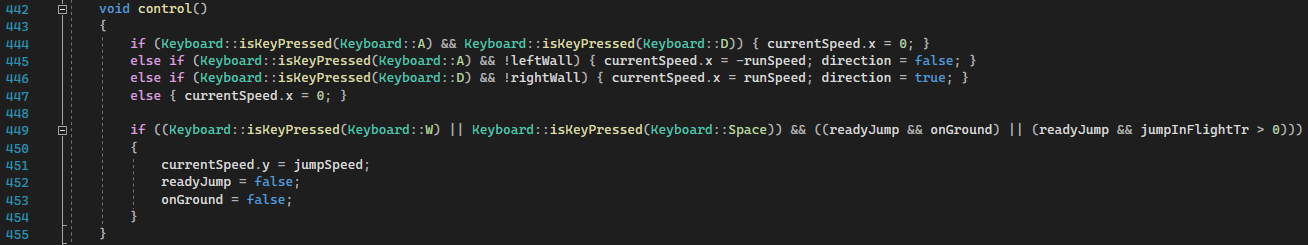


Рисунок 12

Начиная с 444 строки идут проверки на нажатие клавиш клавиатуры. При нажатии на английскую букву A и отсутствие с левой стороны стены, текущая скорость персонажа устанавливается на значение скорости бега назначенную ранее в характеристиках персонажа, но с отрицательным знаком, что обозначает уменьшение X координаты спрайта в окне и, следовательно, его передвижение в левую сторону. Это же верно и для нажатой клавиши D и отсутствия препятствия справа, вследствие чего персонаж перемещается направо. Но все перемещения происходят не в этом методе, а в методе обновления переменных, который будет рассмотрен немного дальше. Помимо текущей скорости персонажа, изменяется еще и его направление, нужное для установки направления анимации. В первом случае false, то есть налево, во втором true, направо. Если нажаты обе клавиши одновременно, текущая скорость устанавливается в нулевое значение, как и в случае если ни одна из клавиш не зажата.

С прыжком, как было сказано ранее, все немного сложнее чем с бегом. Если нажата клавиша W или Space и при этом персонаж готов прыгать и находится на земле, персонаж изменяет свою вертикальную скорость на скорость прыжка, которая имеет фиксированное отрицательное значение. Так как текущая скорость является постоянным значением, если как-либо не ограничить ее, персонаж будет просто не останавливаясь улетать. Ограничение прыжка будет установлено, так же, как и остальное большинство основных ограничений, в методе обновления и будет представлять собой уменьшение вертикальной скорости до нулевой, посредством прибавки постоянной гравитации к ней.

Кроме основного варианта прыжка с земли есть еще и вспомогательный, упрощающий геймплей. Если персонаж готов к прыжку и время с момента, когда он ее покинул не прошло определенный лимит, примерно четверть секунды, то он может прыгнуть в воздухе.

Готовность же к прыжку которая используется в обоих вариантах его допуска, обозначает окончание таймера задержки между прыжками. После прыжка redyJamp становится равной false. Таймер задержки начинает свой отсчет только при нахождении на земле и по истечении времени устанавливает redyJamp в значение true, то есть готов к прыжку.

**3.2. Обновление и интерфейс**

Следующим в очереди на разбор стоит метод обновления update(). Как было сказано ранее именно в нем происходят ключевые изменения характеристик главного героя. Из себя он представляет множество проверок и связанных с этими проверками ограничений, но и не только. Например, изменение положения персонажа в окне происходит каждую итерацию его обновления, но в связи с тем, что обновление кадров завязано на производительности компьютера следует умножать текущую скорость на время, прошедшее между итерациями цикла. Так персонаж будет иметь всегда постоянную скорость, а не рваную, зависящую от случая.



Рисунок 13

К слову о времени, в фрейморке sfml есть свой класс, отвечающий за данную переменную. Он называется Clock и созданный объект этого класса может возвращать прошедшее время с прошлого рестарта в микросекундах, миллисекундах или секундах, в зависимости от требуемой точности. Также стоит заметить, что полученное время делиться на число 800. Данный делитель является скоростью игры, влияющей на все связанные с ней переменные и характеристики, по типу скорости бега или скорости анимации.

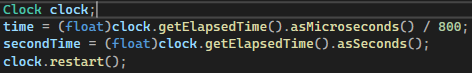


Рисунок 14

Помимо изменения позиции главного героя, каждую итерацию обновляется еще и положение игровой камеры, отслеживающей и непосредственно зависящей от перемещения персонажа, другими словами привязанной к нему.

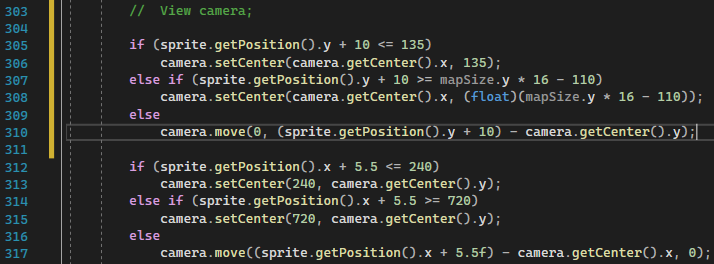


Рисунок 15

В первой последовательности операторов ветвления, 305-310 строки, происходит установление ограничений камеры по Y координате окна. Здесь проверяется достиг ли край камеры, правого или левого края уровня и если же достиг, то устанавливает ограничение на перемещение камеры в правую или левую сторону соответственно стороне достигнутой границы. В противном случае, если края камеры не касаются границ уровня, камера может перемещаться в стороны свободно. Во второй же связке ветвления, 312-317 строки, происходит все тоже самое только по координате X, для верхней и нижней границ окна.

Затрагивая объект игровой камеры, стоит разобрать и класс отвечающий за это. В фреймворке таким классом является View, или же вид. Объект данного класса представляет простой прямоугольник, который играет роль той самой камеры в окне. Все что попадает в этот прямоугольник отображается, остальное же остается за границей видимости. Можно изменять размер прямоугольника и его положение в окне. Точкой отсчета в View по умолчанию является центр прямоугольника, а не как в объекте Sprite его верхний левый угол.

После подбора и установки необходимого размера камеры вида, в данном проекте 480x270 пикселей, нужно установить созданный объект как главный в игровом окне. Делается это при помощи метода окна:

**window.setView(camera);**

После ограничений камеры в коде метода обновления находится проверека на доступность прыжка в воздухе. Эта переменная типа bool устанавливается в значение true в случае если персонаж перестает находиться на земле, но не из-за прыжка. Данная переменная в значении true запускает таймер, до истечения которого персонаж может совершить прыжок, не находясь в это время на земле.

Следом за обработчиком таймеров, идет проверка на падение о которой говорилось в разборе метода управления, именно здесь работает гравитация, оборачивающая силу прыжка вспять и создающая свободное падение. Кроме этого стоит заметить и максимальную скорость падения тела, она устанавливается в конструкторе, как и все остальные переменные.

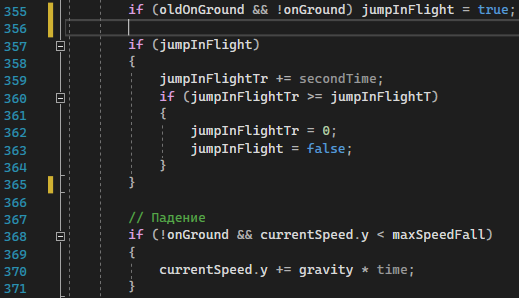


Рисунок 16

Еще немного ниже происходит проверка на выпадение за нижнюю границу уровня. Когда позиция персонажа становится равной или больше чем установленная нижняя граница мира, его текущая скорость по обеим координатам становиться нулевой и неизменяемой, запрет на передвижение. Также запускается таймер, по истечению которого, персонаж восстанавливается на точке возрождения уровня, при этом теряя одну жизнь. Если потерять все 3 жизни, имеющиеся у персонажа, игра будет окончена поражением и ее можно будет начать сначала.

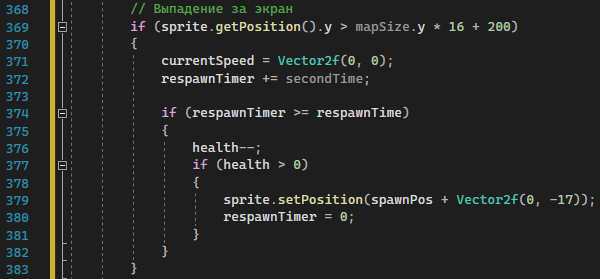


Рисунок 17

Следуя из того, что жизни, как и монеты, должны иметь свой интерфейс, создаются их спрайты и обработчики. В зависимости от количества единиц здоровья персонажа устанавливается нужная текстура в спрайт жизней, и обработчик для этого выглядит вот так:

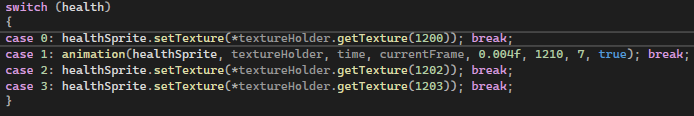


Рисунок 18

Для монет же, помимо изображения требуется еще и текст, который выводится в окно при помощи объекта класса Text.

Созданный интерфейс персонажа в конечном счете имеет еще одну проблему, он не передвигается за камерой и следственно за персонажем, оставаясь на свей стартовой позиции. Для того что бы нужные спрайты оставались в зоне видимости игрока, следует их привязать к персонажу учитывая размер камеры.

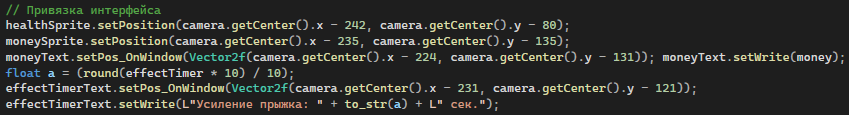


Рисунок 19

Помимо упомянутого интерфейса, в данном коде прописан еще один, таймер длительности эффекта от зелья. Подробнее он будет рассмотрен в главе 6, как дополнительная механика.

Ну и последнее, но не по значению, проверка того, находиться ли персонаж на земле или нет. Эта проверка не изменяет значения переменной, отвечающей за данный параметр, она в зависимости от значения изменяет другие характеристики, по типу направления, влево или вправо, или установки анимации, последовательной смены текстуры в спрайте.



# **Глава 4. Игровая локация**

Игровая локация - это определенная местность, в которой проходит какой-то этап игры. Во время создания игрового мира для простоты разработки его делят на несколько уровней, т.е. определенных зон, в которых будут разворачиваться игровые события.

Это может быть замок с подземельями, фантастический лес, современный город или заброшенный поселок – для создания локации нет ограничений, кроме фантазии и технических возможностей разработчиков. Некоторые игры могут предложить участникам десятки разнообразных локаций, другие же ограничиваются двумя-тремя.

Каждая локация в компьютерной игре представляет собой один изолированный участок мира, который загружается в компьютер игрока. Персонаж может находиться только в одной локации, а для перемещения в другую потребуется загрузка нового пространства игры. Условно локации делятся на два типа:

— **экстерьерные**, представляющие собой открытые пространства – поле, лес, игровую арену, городские улицы, водоемы, открытый космос и т.д.;

— **интерьерные**, где действие происходит в помещении, в качестве которого могут выступать комнаты и коридоры здания, лабиринт пещер или подземелий, канализация, внутренняя структура корабля или космического аппарата.



Рисунок 20

## **4.1. Локация в играх жанра плафтормеры.**

Платформеры — это своеобразный, довольно легко узнаваемый жанр компьютерной игры. Он заключается в том, что у персонажа есть задача — пройти по разным уровням, преодолевая препятствия.

При этом передвигаться предстоит по разным платформам (откуда и пошло название), иногда — лестницам, трубам и прочим. Платформеры в России также известны под названием “бродилки”. Для них характерна рисованная графика, часто — нарочито мультяшная, подчёркивающая искусственный характер происходящего. Главными героями довольно редко становятся люди, пусть и очень условно изображённые. Чаще всего это сказочные персонажи, нередко — антропоморфные. 

Рисунок 21

Платформеры стали популярными в 1980-х годах. Тогда же были сформулированы основные принципы этих игр. Они сводятся к следующему:

* персонаж перемещается по уровням, причём по мере продвижения прохождение вперёд становится всё более сложным;
* у уровней есть секреты, которые необходимо скрыть;
* активно применяются пауэр-апы. Это специальные предметы, благодаря которым у персонажей появляются дополнительные способности. Речь может идти о силовом поле, возможности прыгать или же летать. Однако их действие ограничено во времени;
* простой механизм взаимодействия с предметами. Для этого обычно достаточно напрыгнуть или же соприкоснуться. То есть никаких дополнительных действий совершать не требуется;
* большое количество разнообразных, но достаточно примитивных врагов. Контакт с ними может привести к смерти или же к уменьшению здоровья. В качестве способов справиться с проблемой традиционно предлагается уворачивается, бег, прыжки, а также стрельба (но далеко не всегда);
* появление ловушек и препятствий, которые нужно обходить или же преодолевать с помощью специальных сил;
* движущиеся платформы, падение с которых приводит к потере здоровья или жизни;
* возможность собирать деньги или другие ресурсы, иногда просто очки.

Как было сказано выше, геймплей в игре жанра платформер происходит на локациях или же уровнях, наполненных платформами, активностями и противниками. Поэтому, следующим шагом будет, нахождение информации о том, как построить подобную локацию.

Как оказалось, есть несколько вариантов построения уровней в данном жанре:

* Без разделения на сектора, с произвольно устанавливаемыми на локации объектами.
* При помощи искривления земли и платформ, для создания склонов и углов.
* С разделением на одинаковые сектора, к которым привязываются объекты.

Для проекта был выбран самый простой вариант. Простым же он является как из-за легкости своей реализации, так и из-за относительной простоты оптимизации и построения на его основе физики.

К слову, сектора могут быть разных форм, например, треугольные или шестиугольные. Но для пиксельной игры больше подойдет стандартный квадрат. Карты, построенные на основе таких секторов, называются тайловыми – TileMap. Для их создания и редактирования существует специальный редактор под названием Tiled.

## **4.2. Tiled. Тайловые карты**



Рисунок 22

Работа с бесплатным редактором тайловых карт начинается с его установки на официальном сайте разработчика. После установки и запуска программы пользователя встречает главное меню.

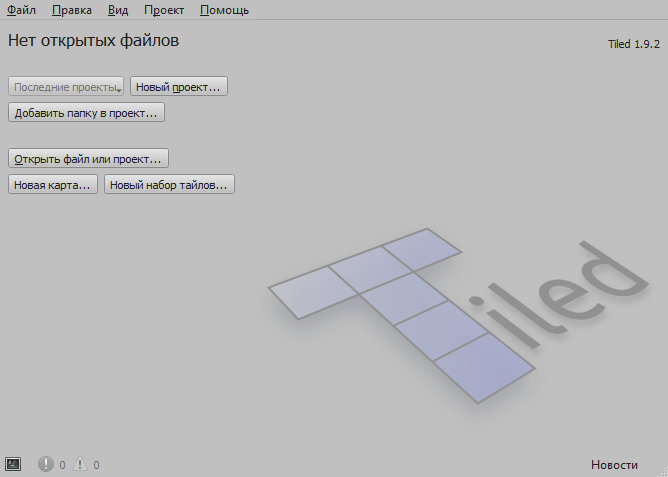


Рисунок 23

Для того чтобы создать первый игровой уровень нужно нажать на кнопку Новая карта… и выбрать нужные настройки в контекстном меню.

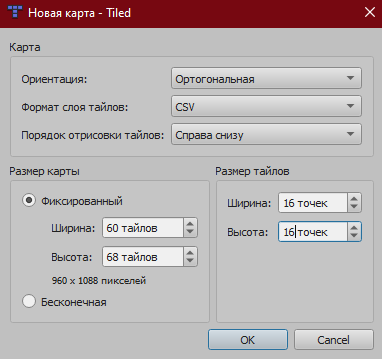


Рисунок 24

Здесь выбираются: начальный размер карты, размер тайла, ориентация, формат слоя тайлов, порядок отрисовки тайлов (поддерживается только для ортогональных карт) и является ли карта бесконечной или нет. Все эти вещи можно изменить позже по мере необходимости, поэтому не важно, чтобы все было правильно с первого раза. Создаваемая для проекта карта первого уровня игры имеет размер 60x68 тайлов и размер одного тайла 16x16 пикселей. Итого, размер уровня в пикселях равен 960 по горизонтали и 1088 повертикали.

После сохранения новой карты будет создана тайловая сетка и на карту будет добавлен начальный тайловый слой. Однако прежде чем будет возможно использовать какие-либо текстуры, нужно добавить набор этих текстур. В меню редактора - Тайл, Новый, Новый набор тайлов… или в правой нижней части окна редактора карты Новый набор тайлов, чтобы открыть одноименное диалоговое окно:

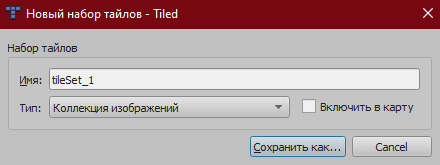


Рисунок 25

В этом меню нужно указать название набора, в данном случае tileSet\_1, а также тип набора, обозначающий каким образом они будут загружаться, коллекцией или одним изображением. Так как в этом проекте важна последовательность тайлов, используется коллекция загружаемых изображений. После сохранения только созданного набора тайлов, следует загрузить в него все необходимые текстуры. Добавление осуществляется простым перетаскиванием нужных файлов из папки компьютера в окно редактирования набора тайлов.

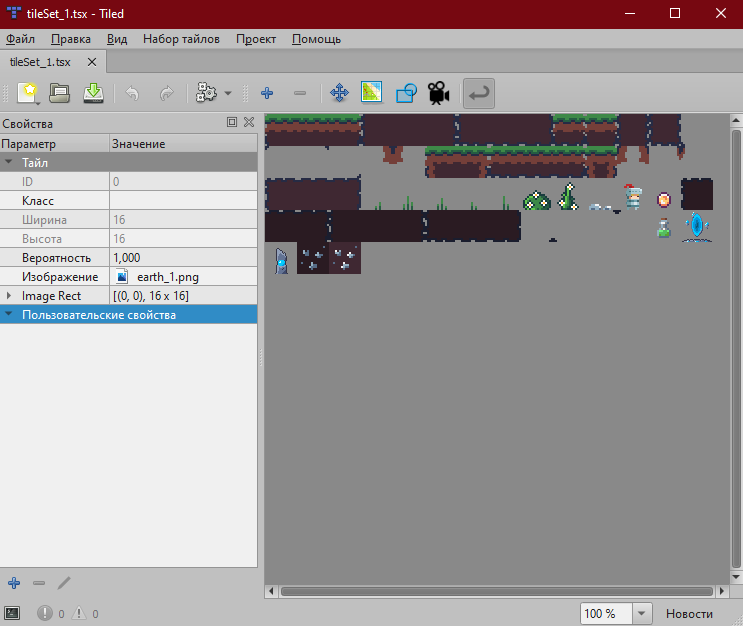


Рисунок 26

В итоге получается заполненный текстурами набор, который можно использовать в редакторе карты, как бы рисуя по клеткам нужными тайлами.

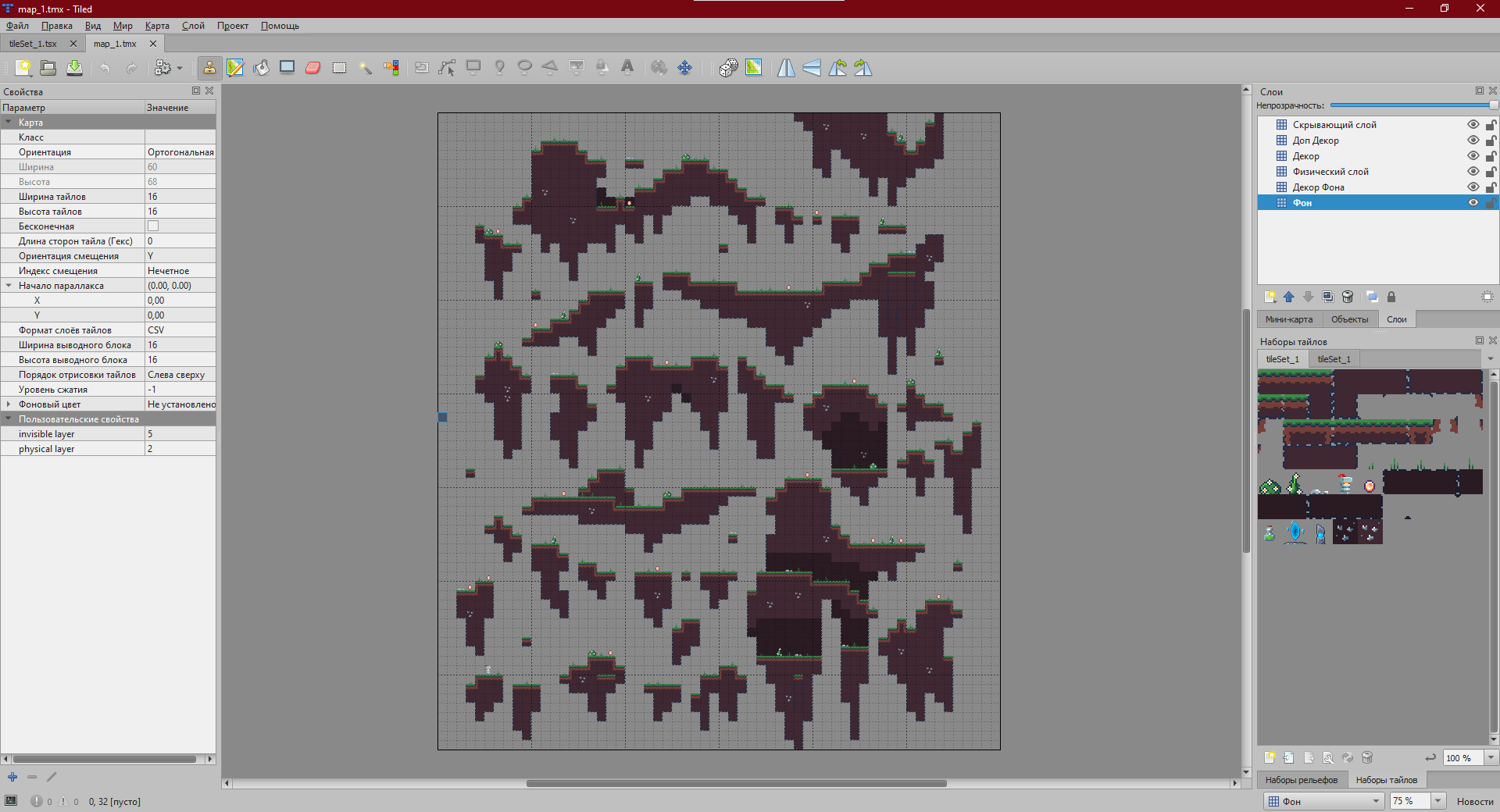


Рисунок 27

В редакторе тайловой карты помимо основного окна рисования и палитры тайлов, присутствует меню разбиения карты на слои и меню свойств, которые могут быть не только стандартными по типу ориентации и размеров, но и пользовательскими, в которых указаны нужные для проекта характеристики карты, физический и прозрачный слои. Прозрачным слоем в данном случае является слой находящийся над персонажем и поэтому перекрывающий его.

## **4.2. RapidXml. Парсинг xml файла**

После сохранения созданных игровых карт, нужно их перенести в окно приложения. Для этого стоит изучить то, во что преобразуется карта из Tiled и как с этим представлением можно работать использую код.

Определить, чем является карта на самом деле можно открыв ее в блокноте или Visual Studio.

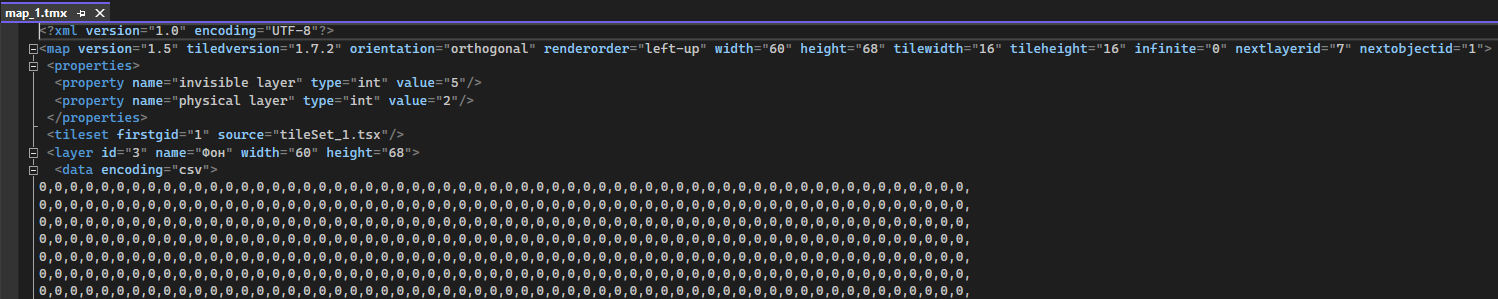


Рисунок 28

Получается, что нарисованная в редакторе карта после сохранения становиться обычным xml файлом, в котором указано все прописанное в Tiled. Сами же слои карты выглядят как последовательность чисел, каждое из которых определяет id текстуры стоящей в данной клетке. Id под номером 0 обозначает пустую клетку без текстуры.

Остается только преобразовать xml в понятный C++ языку вид – этот процесс называется парсингом, а помочь с ним может один из множества фреймворков парсеров. Для данного же проекта был выбран RapidXml как простой и надежный вариант.

После установки и добавления файлов заголовков (.h) в проект для разделения функционала программы, был создан класс карты Map содержащий в себе трехмерную матрицу из стандартной коллекции Vector. Уровни этой коллекции представляют сетку по X и Y осям, а также слои карты. Так же для этого класса были созданы конструктор и функция загрузки из файла.

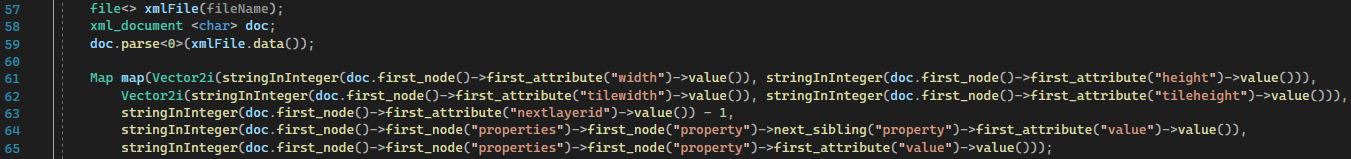


Рисунок 29

Строки 57-59 – указание файла из которого будут извлекаться данные и создание контекста парсинга;

Строки 61-65 – извлечение всех нужных данных карты и помещение их в только созданный класс Map, кроме положения тайлов на карте. Расположение тайлов будет установлено далее, при помощи цикла.

Процесс извлечения данных из xml файла происходит при помощи небольшого количества функций RapidXml. Далее представлен пример кода из проекта, в котором происходит обращение к парсируемому файлу под именованием doc, а затем функция нахождения в файле первого нода:

**doc.first\_node()->first\_attribute("width")->value();**

В xml представлении карты этот нод имеет наименование map и имеет множество дополнительных атрибутов, указанных в той же строке, например, ориентация, ширина и высота карты и даже id следующего слоя. Все их можно достать из файла обратившись к нужному ноду и при помощи функции first\_attribute() и наименования искомого атрибута в его скобках и двойных кавычках. Если же название не указать будет выбран первый атрибут из списка.

В самом конце происходит уточнение того, что нужно получить, а именно value(), содержимое выбранного атрибута, без этой функции пользователь парсера получит название атрибута, которое в данном случае уже известно.

С тем, чтобы получить данные из начала файла, как видно, особых проблем нет, но, чтобы продвинуться дальше нужно понимать логику вложенных нодов и использовать функции требующиеся для этого:

doc.first\_node()->first\_node("properties")->first\_node("property")->next\_sibling("property")->first\_attribute("value")->value()

Так, для того чтобы продвинуться к следующему по глубине ноду в файле, требуется сначала найти входную точку в файл, то есть first\_node(), а затем уже при помощи этой точки пройти к следующей. Это связано с тем что в xml файле может быть только одна входная точка, то есть один нод первого порядка. Так же стоит заметит, что если указать в функции first\_node() название требуемого нода, функция выдаст именно его, а если не указывать, то пользователь получит первый вложенный нод в списке.

Все это замечательно работает с нодами имеющими разные наименования, но если, как в парсируемом файле, несколько нодов одного порядка имеют одинаковое название, используемые функции будут выдавать всегда только первый найденный из нескольких нодов. В таком случае поможет функция next\_sibling(). В данной функции указание наименования нода как и во многих предыдущих необязательно, но предпочтительно, если оно заранее известно.

После получения единичных данных, переходим к довольно крупным их массивам. Для того чтобы реализовать извлечение потребовалось создать два, довольно непростых по реализации цикла fоr, которые проходили свои итерации по уже известным заранее параметрам, высота, ширина и количество слоев.

Первый цикл извлекает числа из файла, переводя крупную строку типа string в массивы int, при этом убирая все пробелы и запятые.

****

Рисунок 30

Второй цикл, используя числовые данные полученные в первом, а так же размеры тайла, загружает в спрайты нужные текстуры, которые дожидались своего времени в классе textureHolder. На данном этапе очень важно, чтобы id текстур в файле и коде совпадали, иначе будут возникать разнообразные ошибки.

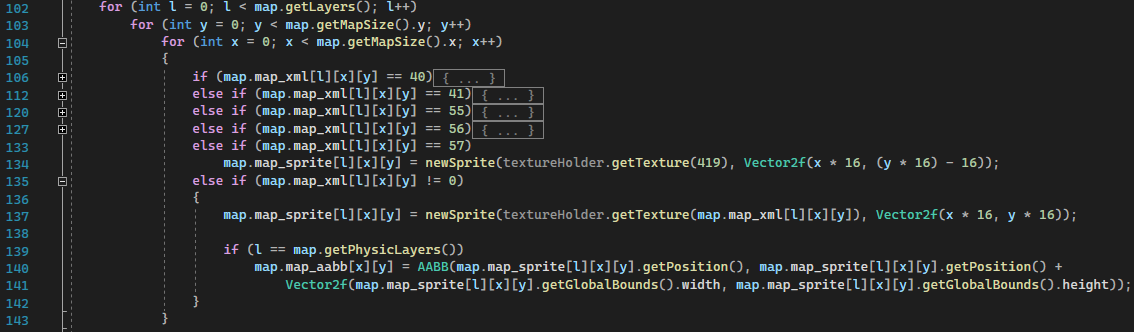
****

Рисунок 31

Помимо прочего, стоит обратить внимание на множество операторов ветвления if в этом цикле, они нужны здесь для того чтобы выделить особые случаи и обработать их иначе чем повсеместные. Например, на карте можно расположить помимо обычных тайлов окружения еще и интерактивные, монеты, зелья и т.д. они требуют для себя сторонние коллекции и поэтому отделяются от обычных изображений земли. Или же, на все той же карте можно расположить точку появления игрока на уровне, что будет несомненно гораздо удобнее чем постоянное изменение его положения через код, это так же будет выделено в данных и обработано иначе.

После всех проверок на свою исключительность, а также после окончательной проверки на пустоту, текстура спокойно помещается в выделенное для нее место в коллекции. Если же выбранный тайл входит в физический слой он получает еще и хитбокс. Речь о том что это такое будет вестись в следующей главе.

# **Глава 5. Физика**

# **Глава 6. Дополнительные механики**

# **Заключение**

# **Список литературы**

1. Грегори Джейсон, Игровой движок. Программирование и внутреннее устройство. Третье издание / Перевод: Сивченко О. Ю., Черников С.В. Питер, 2022г.–1136с.

# **Листинг**

# **Экономическая часть**