МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Кафедра программного обеспечения систем радиоэлектронной аппаратуры

КУРСОВАЯ РАБОТА

по дисциплине «Методы и стандарты программирования»

на тему: «Игра в жанре rogue-like»

Обучающийся _____ _____
Подпись *Фамилия Имя Отчество*

Шифр 23К0071

Группа КМБО-05-23

Руководитель работы _____ _____
Подпись *Фамилия Имя Отчество*

Москва 2022

Содержание

1. Введение

Темой курсовой работы является создание игры в жанре "rogue-like". Цель игры - пройти возможно большее количество уровней, сгенерированных случайным образом. Игроку предстоит сражаться на каждом уровне с противниками, набор которых также сгенерированный случайно. Наконец, игрок может подбирать случайно появляющиеся на уровнях предметы.

Для осуществления проекта, необходимо было проанализировать механику игры и выбрать инструменты для её создания.

В данном отчете описан процесс создания игры и руководство пользователя.

2. Техническое задание

2.1. Общее описание

Проект представляет собой игру в жанре rogue-like. Устройство игрового процесса следующее: игрок появляется в структурно генерируемом мире (это значит, к примеру, что заранее определено некоторое количество локаций, созданных руками разработчика, однако на каждом уровне набор этих локаций выбирается случайным образом, равно как и связи между ними, в смысле путей между локациями). В каждой комнате встречаются враги, и цель игрока – очистить комнату, чтобы пройти дальше. В некоторых комнатах могут встретиться предметы, усиливающие игрока. Соответственно, количество и состав врагов, предметы, встречающиеся в комнатах с наградами и усиливающие игрока – все это также генерируется случайным образом. При прохождении каждого уровня игра становится сложнее. Основная цель игры – пройти как можно большее количество уровней и набрать как можно большее количество очков.

2.2. Описание графического пользовательского интерфейса

В игре присутствует несколько экранов, каждый из которых отвечает своим целям:

- Первым делом, игрока встречает главное меню. Меню представляет собой набор из нескольких кнопок, расположенных горизонтально: “Новая игра”, “Настройки”, “Выход”. Кнопка “Новая игра” позволяет при нажатии запустить новую игру или возобновить приостановленную; окно “Настройки” позволяет перейти в меню настроек; кнопка “Выход” позволяет выйти из игры;
- Меню настроек представляет собой набор различных параметров, которые игрок может изменять, а именно: разрешение экрана и громкость звука;
- Игровой экран представляет собой окно, непосредственно в котором отображается игра, со всеми ее объектами. Помимо этого, игровой экран имеет head-up display, позволяющий просмотреть информацию о текущем счете, количестве очков здоровья у игрока, проценте заряда особых способностей игрока;
- Миникарта. При нажатии на определенную клавишу, миникарта раскрывается, позволяя игроку увидеть карту уровня и ориентироваться внутри него.

2.3. Описание игровых сущностей

- Главный герой игры – сущность, управляемая игроком с помощью мыши и клавиатуры. Со старта главный герой умеет стрелять в направлении, указываемом мышью; выполнять перекат, временно уходя в неуязвимость от вражеского урона; выполнять особое умение, заряжающееся с нанесением урона – при активации зона вокруг главного героя очищается от вражеских снарядов, а враги получают некоторый урон внутри этой зоны;
- Враги представляют собой сущности, задача которых – усложнить жизнь игроку. Они могут наносить урон игроку. Существует 5 различных разновидностей врагов внутри игры:
    - Обычный враг стреляет в направлении игрока одиночным выстрелом;
    - Танк наносит урон только вблизи, однако этот урон больше, чем у обычного врага. Также, здоровье танка больше, однако его скорость меньше обычной;
    - Волшебник запускает в главного героя огненные шары, которые проходят через стены и взрываются. На месте взрыва возникает область, проходя через которую игрок получает урон;
    - Снайпер, в отличии от обычного врага, стреляет лазером, моментально достигающим цели и разделяющий комнату собой. Проходя через лазер, игрок также получает урон;
    - Волки – быстрые враги, бьющие главного героя в ближнем бою. Обладают наименьшим запасом здоровья из всех врагов;
- Предметы – сущности, появляющиеся в локациях с наградой. Игрок может поднимать предметы, тем самым усиливая себя. Эффект от нескольких одинаковых предметов суммируется. В игре

существует 13 различных предметов, разделенных на три категории:

- Обычные предметы встречаются чаще всего. Существуют предметы, которые увеличивают скорость перемещения, скорость стрельбы, урон, шанс оглушить противника (оглушенный противник не может двигаться и стрелять). Также, есть два предмета, которые позволяют увеличить максимальное количество очков здоровья и восстановить их некоторое количество. Всего шесть обычных предметов;
- Необычные предметы встречаются реже, однако их действие на игрока более существенно. Существуют предметы, которые позволяют отравить врага с некоторым шансом при нанесении урона; заменить обычный выстрел на лазер; запускать огненный шар с некоторой вероятностью при выстреле; получить способность вампиризма – восстанавливать здоровье; увеличить скорость заряда особой способности, скорость заряда переката. Всего шесть необычных предметов;
- Особые предметы встречаются наиболее редко, однако один особый предмет вносит очень большой вклад в игру. Существует один особый предмет, дающий дополнительную жизнь.

3. Отчет о разработке

Программа разбита на несколько модулей. Под словом "модуль" подразумевается набор файлов реализации и соответствующих заголовочных файлов, которые объявляют классы и функции, решающие общую задачу. Для создания кода для сборки используется CMake, поэтому каждый модуль описывается библиотекой, определяемой в файле CMakeLists.txt. Каждый модуль отвечает за определенный набор задач, которые будут описаны далее:

1. gamegui - модуль, отвечающий за создание и управление графическим пользовательским интерфейсом внутри игры. Используется для создания игрового меню.
2. base - модуль, содержащий базовые классы программы: Game, MapManager и Settings.
3. entities - модуль, содержащий классы всех внутриигровых сущностей: плиток, из которых формируется карта; противников; предметов, которые игрок может подбирать; снарядов, используемых игроком и врагами для собственной атаки.
4. sound - модуль, содержащий класс для проигрывания звуков внутри игры
5. vecmath - вспомогательный модуль, содержащий функции для работы с векторами

Рассмотрим иерархию классов каждого модуля по отдельности.

3.1. Иерархия классов модуля base

Game - основной класс игры, объединяющий внутри себя все остальные классы и функции, и реализующий непосредственно игру. Класс game реализует игровой цикл (_gameloop(), _ingameHandling()), обработку действий игрока (_inputHandling()), создание и отрисовку пользовательского интерфейса (функции с префиксом _set: _setMenusWindows(), _setSettingsWindow(), и т. д.).

MapManager - класс, отвечающий за создание и управление картами внутри игры. При создании объекта класса последовательно читаются текстовые файлы из папки "resources/maps" и на основе информации внутри них конструируется эталонный набор комнат. Из случайно выбранных из этого набора комнат функцией generateNewLevel() создается уровень. Функцией getRoom можно получить доступ к определенной комнате уровня.

Map - класс, содержащий внутри себя информацию об уровне: массив блоков, формирующих комнату; массивы, которые формируют граф, необходимый врагам для поиска пути к игроку и для передвижения (_mapCoords, _adjList); массив предметов; различные состояния комнаты (например, зачищена ли комната от врагов).

Settings - класс, реализующий изменение и сохранение настроек. Инкапсулирует работу с файлом "resources/settings.txt", который хранит настройки игры: громкость звука и размер экрана.

3.2. Описание классов модуля gamegui

GuiObject - абстрактный класс, определяющий интерфейс для всех остальных объектов. Объект должен реализовать функцию отрисовки себя, изменения размера и позиции, а также функцию, вызываемую при активации объекта.

AbstractPacker - класс, являющийся основой для остальных упаковщиков. Упаковщик - контейнер, содержащий внутри себя остальные объекты GuiObject, возможно, другие упаковщики. Вызов любого метода класса GuiObject приводит к вызову данного метода для всех объектов, которые содержит упаковщик. Так возможно создавать окна со сложным дизайном и структурировать код графической составляющей программы.

AbsolutePacker - класс упаковщика, который размещает объекты внутри себя согласно переданным координатам. Начало отсчета - левый верхний угол упаковщика.

GridPacker - класс упаковщика, выравнивающий объекты внутри себя по сетке.

Button - класс кнопки. Функцией setFunc устанавливается замыкание, вызывающееся при нажатии на кнопку. Также, существует большие возможности для конфигурации кнопки функциями-сеттерами.

Label - класс, реализующий строку, которая выводится на экран. При создании объекта класса передается ссылка на объект std::string, и его содержимое выводится на экран. При изменении строки, меняется и выводимое изображение.

Image - класс, содержащий внутри себя некоторое изображение, выводимое на экран.

Slider - класс, реализующий объект слайдера. Слайдер представляет собой ползунок, перемещающийся вдоль прямоугольника. При перемещении ползунка вызывается замыкание, переданное в функцию setFunc. Замыкание должно принимать в качестве аргумента число типа double - отношение расстояния от ползунка до левого края к общей длине.

Диаграмма классов модуля представлена на рисунке 1. Здесь и далее принимаются некоторые соглашения, связанные со значением графических знаков на диаграммах. Синими прямоугольниками отмечены абстрактные классы, черными прямоугольниками – обычные. Стрелки обозначают отношение “Родительский класс – класс-наследник”.

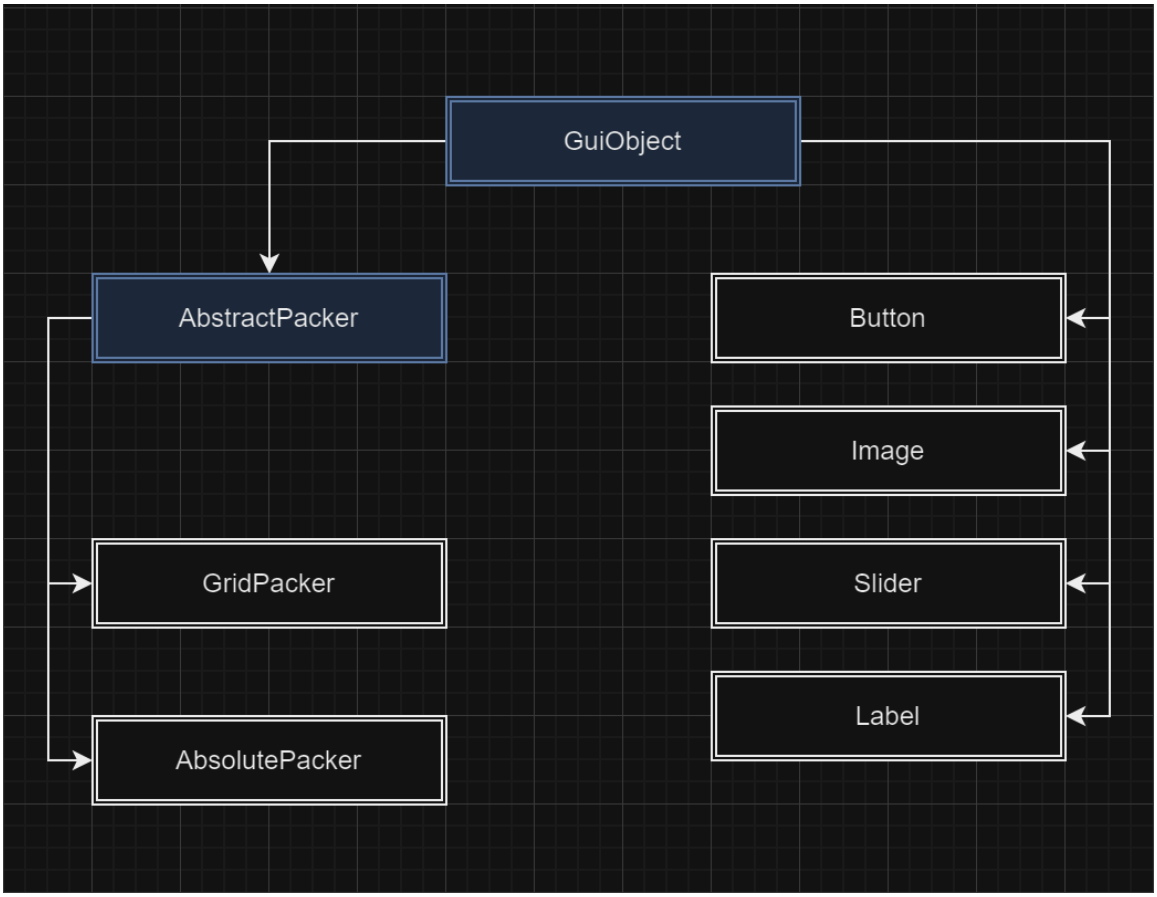

Рисунок 1 – Диаграмма классов модуля “gamegui”

3.3. Описание класса модуля entities

Entity - абстрактный класс, базовый для всех сущностей. Определяет интерфейс, который необходимо реализовать каждой сущности в игре. enum EntityType - перечисляемый тип, определяющий все возможные виды сущностей внутри игры. Каждая сущность хранит внутри себя поле типа EntityType, определяющее тип данного объекта.

Movable - абстрактный класс, определяющий интерфейс для всех сущностей, которые способны двигаться: игрок, враги, пули и т. д.

Gamer - класс, реализующий сущность, управляемую игроком.

AbstractEnemy - абстрактный клас противника. Внутри себя реализует алгоритм поиска пути к игроку. Результат работы этого алгоритма - набор координат, по которым должен двигаться противник, чтобы достичь игрока. Алгоритмы непосредственно движения реализуются в классах-наследниках.

Striker, Sniper, Wolf, Bigboy, Wizard - классы врагов, описанных в техническом задании.

AbstractShot - абстракный класс, предоставляющий интерфейс для всех возможных видов снарядов в игре. Внутри себя содержит указатель на объект игрока, что позволяет реализовать некоторые механики игры, например, вампиризм.

Shot - класс базового снаряда: пуля, вылетающая с большой скоростью.

Lazer - класс лазера. Заряжается некоторое время, после чего начинает наносить урон, пока не исчезнет. Лазер наносит урон всем сущностям, прошедшим через него.

Fireball - класс магического снаряда. Летит медленней обычной пули, проходит через стены, при столкновении с сущностью или по истечении определенного времени взрывается и воспламеняет зону карты. Попавшие в эту зону получают продолжительный урон.

UltCharge - класс особой способности игрока.

Item - класс внутриигровых предметов. Каждый описан в пункте "Техническое задание".

Диаграмма классов модуля представлена на рисунке 2.

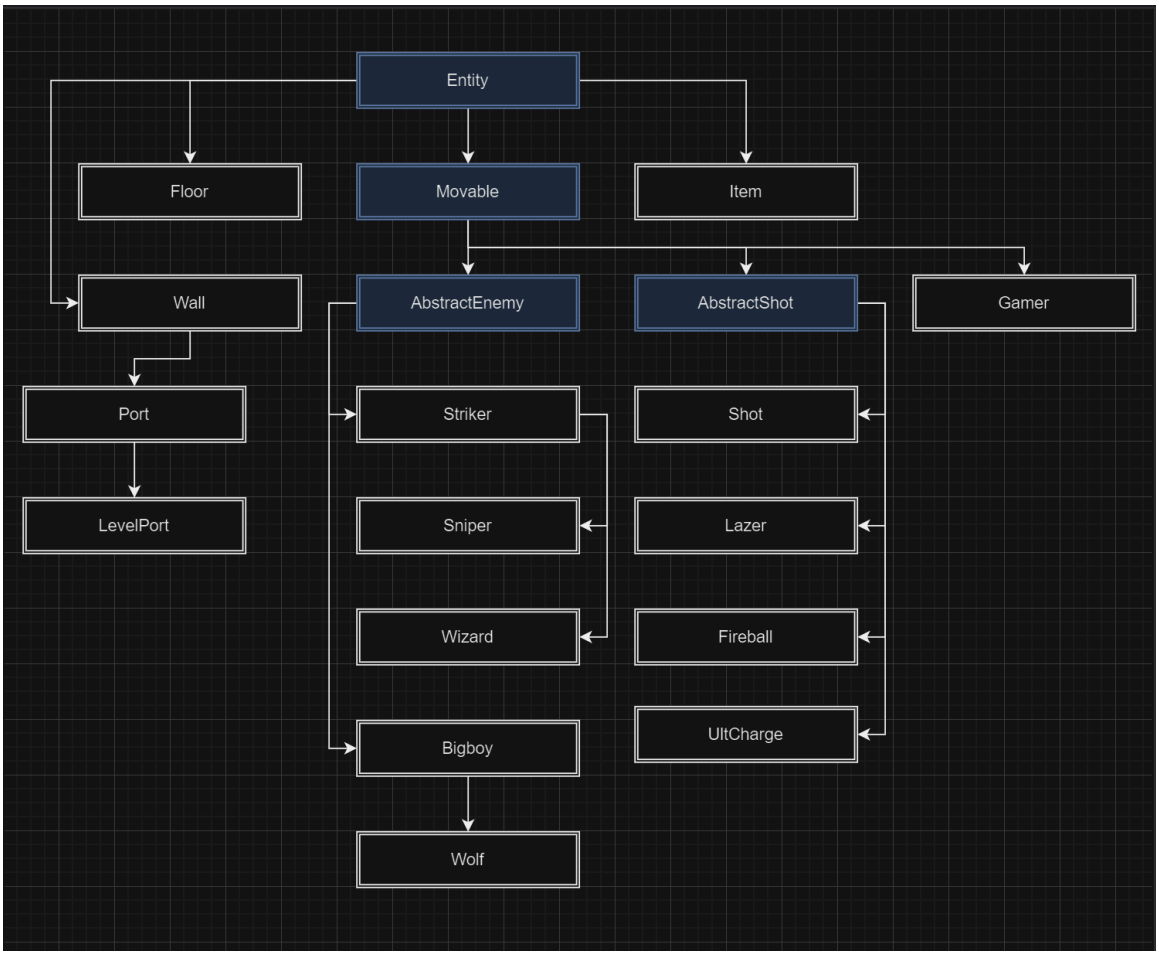

Рисунок 2 – диаграмма классов модуля “entities”

3.4. Описание вспомогательных модулей sound и vecmath

SoundManager - класс, позволяющий сущностям проигрывать звуки. Объект класса хранится как статическое поле класса Movable.

vecmath.cpp - содержит функции для работы с векторами и общематематические функции и константы.

3.5. Использованные алгоритмы

При реализации искусственного интеллекта врагов был применен алгоритм обхода графа в ширину (Breath-first search, BFS). Для каждой комнаты классом MapManager рассчитывается граф – сетка, покрывающая все плитки карты, по которым можно передвигаться. Так, например, по стенам нельзя передвигаться, и они не учитываются при расчете графа.

После этого, граф сохраняется в статическое поле класса AbstractEnemy. Над графом исполняется BFS для отыскания кратчайшего пути до определенной точки комнаты (не обязательно до позиции игрока: стреляющие враги держат дистанцию между собой и игроком, поэтому если игрок находится слишком близко ко врагу, то этот враг будет убегать от игрока в случайно выбранную точку на карте – “точку паники”).

Также, BFS используется при генерации уровня: во время создания уровня при помощи этого алгоритма ищется наиболее удаленная комната от той, в которой появляется игрок. Таким образом, там помещается телепорт на следующий уровень.

4. Руководство по сборке

Поскольку SFML – кроссплатформенная библиотека, игру возможно собрать и для Linux, и для Windows. Рассмотрим сначала руководство по сборке для Linux.

Необходимые зависимости для сборки игры для Linux:

- SFML
- CMake
- Любая система сборки (make, ninja, и т. д.)
- Любой компилятор для языка C++ (gcc, clang, и т. д.)

Для начала устанавливаем все необходимые зависимости. Для многих популярных дистрибутивов, всё вышеперечисленное либо доступно в системе изначально, либо доступно для установки официальным пакетным менеджером дистрибутива.

После установки зависимостей создаем внутри папки проекта папку, где будут храниться файлы сборки и запускаем команду “*cmake* ..”. Далее запускаем систему сборки, например, командой “*make*”. После окончания сборки копируем папку “resources” из папки проекта в папку сборки и запускаем файл “main”.

Процесс сборки игры для Windows принципиально не отличается от вышеизложенного, однако появляются некоторые нюансы, которые следует учитывать. Во-первых, набор систем сборки и компиляторов отличается. Так, автором курсовой использовался набор разработки MinGW. Во-вторых, на этапе запуска CMake потребуется обозначить некоторые флаги:

- -DCMAKE_PREFIX_PATH= - сразу после в кавычках следует путь расположения файлов SFML. Например, флаг может быть указан так:

  -DCMAKE_PREFIX_PATH="C:\Program Files (x86)\SFML-2.6.1";
- -G – сразу после в кавычках следует система сборки, для которой CMake создаст необходимые инструкции. Поскольку автор

курсовой пользовался MinGW, при сборке указывался флаг: -G"MinGW Makefiles";

Наконец, после сборки для запуска игры может потребоваться переместить некоторые DLL-файлы библиотек в папку сборки: некоторые бинарные файлы стандартной библиотеки и файлы библиотеки SFML. После сборки не следует забывать, что в папку сборки нужно скопировать файл “resources”.

5. Руководство пользователя

При запуске игры пользователь попадает в главное меню. Назначение кнопок описано в разделе 2.2. Стоит отметить, что изменение настроек вступает в силу только после перезапуска игры, о чем сообщает пользователю текст в меню настроек.

Управление в игре следующее:

- Поворот мыши – поворот внутри игры в сторону курсора пользователя,
- Зажатие левой кнопки мыши – стрельба,
- Нажатие кнопок “W”, “A”, “S”, “D” – управление персонажем внутри игры, движение вверх, влево, вниз и вправо соответственно,
- Нажатие “Escape” – пауза. При нажатии кнопки “New game” приостановленная игра продолжается,
- Нажатие “Space” – выполнение переката,
- Нажатие “E” – активация особой способности
- Нажатие “Tab” – активация миникарты

Внешний вид игрового окна приведен на рисунке 3.

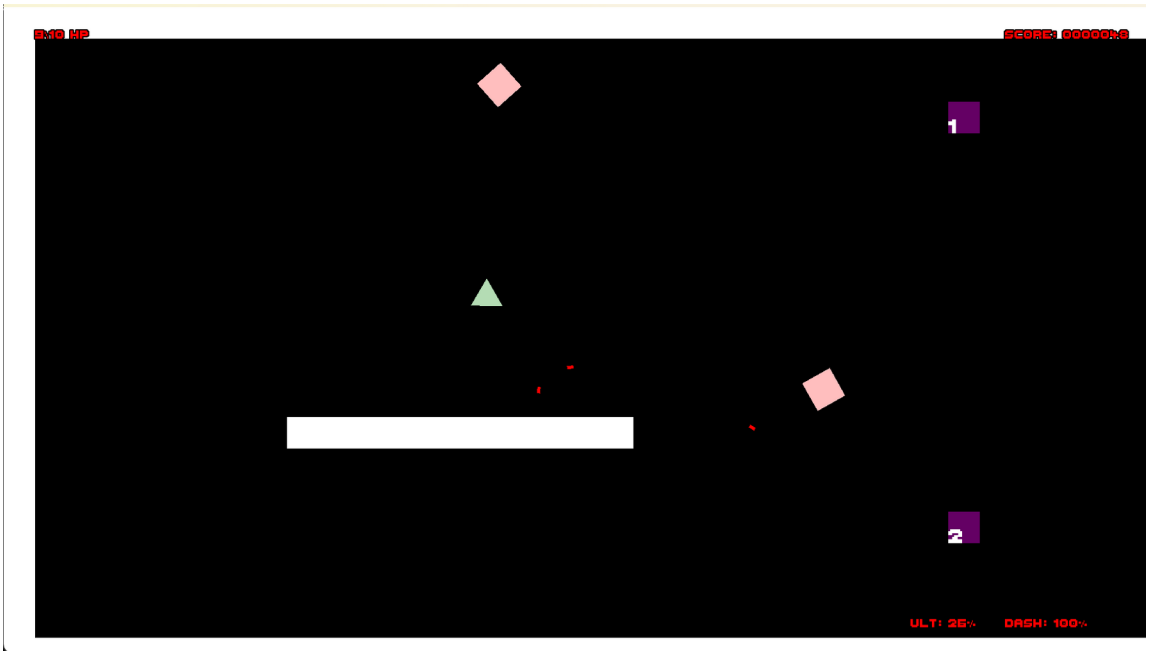

Рисунок 3 – пример игрового процесса

При нажатии на “Tab” демонстрируется миникарта, пример которой изображен на рисунке 4. Квадраты с номерами – пронумерованные комнаты. Зеленым цветом отмечена комната, в которой находится игрок. Также внутри каждой комнаты находятся фиолетовые квадраты – телепорты, номера на которых соответствуют номеру комнаты на миникарте. Таким образом, игрок может осуществлять навигацию по уровню. Темным фиолетовым на миникарте отмечены комнаты еще не зачищенные, ярким фиолетовым – уже зачищенные комнаты. Желтым отмечена комната с телепортом на следующий уровень. Комнаты, у которых номер написан желтым цветом, – это комнаты с предметами. Если между комнатами на миникарте есть связь (серый прямоугольник с концами в двух комнатах), то это означает, что через телепорты можно попасть из одной комнаты в другую, и наоборот.

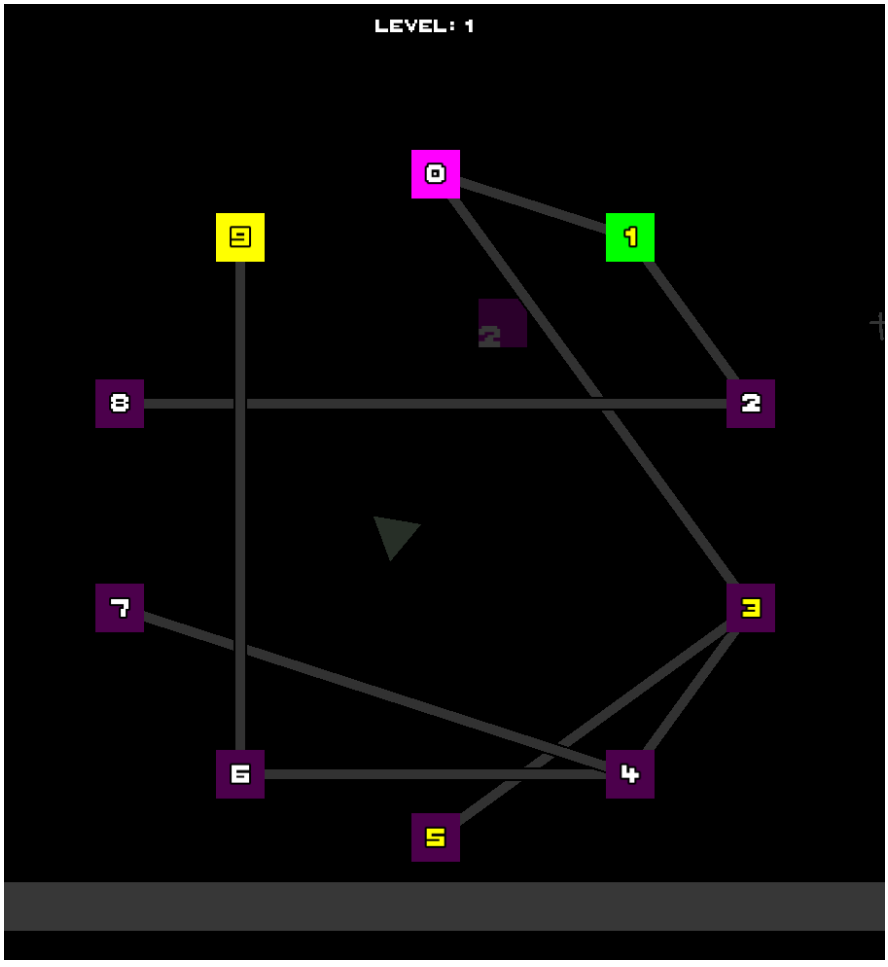

Рисунок 4 – пример миникарты уровня

Как уже отмечалось ранее, на некоторых уровнях есть предметы. Далее следует сопоставление предметов и их изображений внутри игры:

- Ботинок – увеличение скорости передвижения,
- Кинжал – увеличение урона,
- Банка газировки – увеличение скорости стрельбы,
- Красное сердце – увеличение максимального количества очков здоровья,
- Синее сердце – восстановление очков здоровья,
- Молот – шанс оглушить врага при нанесении урона,
- Маска – уменьшение времени восстановления особого навыка,
- Книга с надписью “52” – уменьшение времени восстановления переката,
- Серп – вампиризм,
- Оранжевый шар – шанс запустить огненный шар во врага при выстреле,
- Капля яда – шанс отравить врага при попадании,
- Алмаз – замена основного оружия на лазер,
- Надпись “+1” – дополнительная жизнь.

После того, как очки здоровья закончились, а дополнительных жизней не осталось, игра завершается, демонстрируя игроку количество набранных очков.

6. Заключение

В ходе выполнения курсовой работы мною были приобретены практические навыки по созданию программного продукта на языке C++: изучена система сборки CMake; получено глубокое понимание того, как устроено объектно-ориентированное программирование в C++; изучены и использованы многие механизмы “современного” C++, в том числе и стандарта C++17, такие как std::optional, std::filesystem; закреплены знания о работе с Git, полученные в рамках курса “Программирование в задачах радиолокации”. Также, был приобретен опыт разработки компьютерных игр, изучены основные концепции, применяемые при разработке игр, получен навык работы с SFML. Наконец, были закреплены знания, полученные в рамках курса “Методы и стандарты программирования”.

7. Список использованной литературы

1. Кормен, Томас Х. и др. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. – М. : ООО “И. Д. Вильямс”, 2013. – 1328 с.

2. Мейерс, Скотт. Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14. : Пер. с англ. – М. : ООО “И. Д. Вильямс”, 2016. – 304 с.

3. Документация системы сборки CMake: https://cmake.org/cmake/help/latest/

4. Обучение работы с системой сборки CMake: https://cmake.org/cmake/help/latest/guide/tutorial/

5. Документация для библиотеки SFML: https://www.sfml-dev.org/documentation/2.6.2/

6. Документация языка C++: https://en.cppreference.com/w/