## **Введение**

На преддипломную практику была поставлена задача – разработка игрового приложения “OnionFree”.

Целью преддипломной практики является создания игрового приложения в жанре Roguelite, для развлечения и отдыха.

Для работы с программой выход в интернет не нужен: она работает с файлами, содержащими все необходимое для использования.

Приведём краткое описание разделов пояснительной записки.

Первый раздел «Структура и общая технологическая схема производства, характеристики основных видов продукции» посвящен описанию структуры предприятия и характеристике основных видов производимой или выпускаемой продукции (услуг).

Во втором разделе «Описание структуры ВЦ, АСУП и производственного отдела, где проходит практика» приводится описание структуры отдела, в котором проходит практика.

Третий раздел «Должностные обязанности оператора ЭВМ, техника-программиста, инженера-программиста». В нем описываются непосредственные обязанности оператора ЭВМ, техника-программиста и инженера-программиста.

В разделе «Анализ задачи» вы сможете ознакомиться с постановкой задачи, включающей в себя: анализ предметной области поставленной задачи. Также в этом разделе вы сможете узнать о том, как данная задача решается в настоящее время. В подразделе «Инструменты разработки» будет рассмотрена среда, в которой создается данный проект. Здесь также будут установлены минимальные и оптимальные требования к аппаратным характеристикам, обеспечивающим правильное функционирование поставленной задачей.

В разделе «Проектирование задачи» будут рассмотрены основные аспекты разработки web-ресурса. Здесь можно будет узнать о коллекции, которая содержит в себе список элементов и правил, подходящих под стилистику данного сайта и которым дизайнеры и разработчики должны следовать.

«Реализация задачи» – раздел отчета по практике, в котором описываются все элементы и объекты, которые будут использованы при реализации данного приложения. В этом разделе будут четко описаны функции пользователя и их структура предприятия и характеристике основных видов производимой или выпускаемой продукции (услуг).

Седьмой раздел – «Тестирование». В нем будет описано полное и функциональное тестирование данной программы, т.е. будет протестирован каждый пункт меню, каждая операция, которая выполняется web-ресурсом. Будут смоделированы все возможные действия пользователя при работе с web-ресурсом, начиная от входа на сайт заканчивая закрытием вкладки.

В разделе «Применение» будет описано назначение и область применения программы.

«Заключение» будет содержать краткую формулировку задачи, результаты проделанной работы, описание степени автоматизации процессов на различных этапах разработки.

В разделе «Список используемых источников» будет приведен список используемых при разработке источников.

В приложении «А» приведен листинг программы.

**1 Структура и общая технологическая схема производства, характеристики основных видов продукции**

ЧП «БухУс» – бухгалтерские услуги для ИП и малого бизнеса:

* предоставим персонального бухгалтера с опытом работы;
* поможем в решении проблем с контролирующими органами;
* гарантируем качество, оперативность и конфиденциальность.

Данное предприятие предоставляет ряд разного рода услуг, связанные с бухгалтерией для ИП и малого бизнеса. Например, такие услуги, как востановление бухгалтерского и налогового учета, подготовка к проверке закрытию ИМНС, ФСЗН, Белгосстрах, электронное декларирование, электронные света-фактуры, составление штатного расписания и тому подобные услуги.

Также ЧП «БухУс» предоставляет услуги введения таких отраслей, как оптовая и розничная торговля, услуги (строительство, транспорт и другое), производство, ВЭД (экспорт, импорт).

**3 Должностные обязанности оператора ЭВМ, техника-программиста, инженера-программиста**

Должностные обязанности оператора ЭВМ:

* обработка информации на ЭВМ;
* ввод информации в ЭВМ с носителей информации и каналов связи;
* контроль носителей информации;
* работа с электронными таблицами;
* выполнение работ по тестированию антивирусными программами и всех файлов.

Должностные обязанности техника-программиста:

* выполнять работу по обеспечению механизированной и автоматизированной обработки, поступающей в вычислительный (информационно-вычислительный) центр (ВЦ, ИВЦ) информации, разработки технологии решения экономических и других задач производственного и научно-исследовательского характера;
* принимать участие в проектировании систем обработки данных и систем математического обеспечения машины;
* выполнять подготовительные операции, связанные с осуществлением вычислительного процесса, вести наблюдение за работой машин;
* составлять простые схемы технологического процесса обработки информации, алгоритмы решения задач, схемы коммутации, макеты, рабочие инструкции и необходимые пояснения к ним;
* разрабатывать программы решения простых задач, проводить их отладку и экспериментальную проверку отдельных этапов работ;
* участвовать в выполнении различных операций технологического процесса обработки информации (прием и контроль входной информации, подготовка исходных данных, обработка информации, выпуск исходящей документации и передача ее заказчику);
* вести учет использования машинного времени, объемов выполненных работ;
* оказывать содействие и сотрудничать с нанимателем в деле обеспечения здоровых и безопасных условий труда.

Должностные обязанности инженера-программиста:

* на основе анализа математических моделей и алгоритмов решения научных, прикладных экономических и других задач разрабатывать программы, обеспечивающие возможность выполнения средствами вычислительной техники алгоритма и поставленной задачи;
* Участвовать в выборе языка программирования для описания алгоритмов и структур данных;
* Разрабатывать, отлаживать, анализировать и оптимизировать программный код на основе готовых спецификаций;
* Интегрировать программные компоненты;
* Проводить проверку программ на основе логического анализа;
* Проводить отладку разработанных программ, корректировать их в процессе стабилизации и сопровождения;
* выполнять работу по унификации процессов разработки программ;
* разрабатывать и внедрять методы автоматизации программирования;
* Участвовать в сопровождении программного обеспечения;
* Вести и представлять установленную отчетность.

## **4 Анализ задачи**

## **4.1 Анализ предметной области**

Темой данного курсового проекта является разработка игрового приложения «OnionFree».

Разрабатываемый программный продукт должен позволять выполнить следующие действия:

* Работать без сбоев и ошибок

В разработанной программе будут использоваться один вид данных. К выходной информации можно отнести результат о прогрессе рекорда.

Игра - это концентрация на процессе, игровое приложение — на результате. Игровое приложение строится на внешней мотивации: дети играют, чтобы получить виртуальные бонусы. Видеоигра строится на внутренней мотивации: дети играют не только ради наград, но и ради самой игры.

Виды игровых приложений:

* головоломки;
* настольные интеллектуальные игры в компьютерном варианте;
* спортивные игры;
* военные игры;
* авантюрные игры;
* конвейерные игры;
* учебно-технологические игры;
* аркадные.

Приложение, которое будет разрабатываться, будет относится к виду военные, аркадные.

## **4.2 Инструменты разработки**

Для оптимального функционирования разрабатываемого приложения необходимо наличие у компьютера следующих параметров:

− 1.8 GHz процессор (или более быстрый);

− 1 GB RAM;

− от 75 MB доступного места на жестком диске.

Итак, для разработки программы необходимы:

1. Visual Studio 2019 – главная среда разработки программы, в которой прописывается код и создаются формы;
2. Google-браузер – для нахождения информации, картинок;
3. Word 2016 – для написания пояснительной записки;
4. Adobe Photoshop CC – для редактирования фотографий;
5. Операционная система Windows 10.

Сетевая поддержка – отсутствует.

**4.3 Диаграмма вариантов использования**

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Основные элементы диаграммы - участник (actor) и прецедент (вариант). Участник — это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Участником может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности. Графически участник изображается «человечком».

Прецедент (use case) - описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому участником результату. Прецедент представляет поведение сущности, описывая взаимодействие между участниками и системой. Прецедент не показывает, “как” достигается некоторый результат, а только “что” именно выполняется. Прецеденты обозначаются очень простым образом - в виде эллипса, внутри которого указано его название.

На диаграмме будет представлен один актёр: пользователь. Диаграмма вариантов использования представлена на рисунке 1.

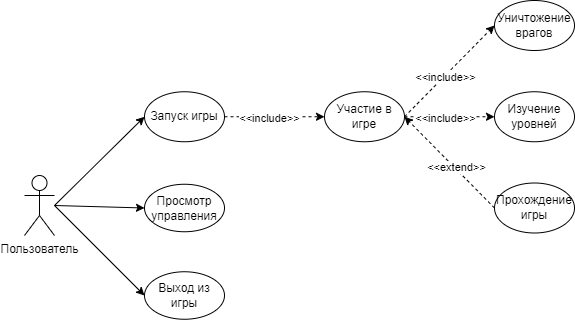


Рисунок 1 – Диаграмма вариантов использования

**4.4 Модель данных**

Моделирование данных — это создание визуального представления обо всей информационной системе либо ее части. Цель в том, чтобы проиллюстрировать типы данных, которые используются и хранятся в системе, отношения между этими типами данных, способы группировки и организации данных, их форматы и атрибуты.

Модель данных для сайта, выбранного на дипломное проектирование, представлена на рисунке 2.

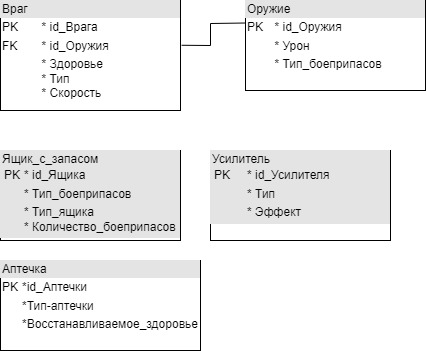


Рисунок 2 – Модель данных

**4.5 Постановка задачи**

Темой данной преддипломной практики является разработка игрового приложения «OnionFree».

Разрабатываемый программный продукт должен позволять выполнить следующие действия:

* Работать без сбоев и ошибок
* Выполнять все функции, которые в него вложены

В разработанной программе будут использоваться один вид данных. К выходной информации можно отнести результат о прогрессе рекорда.

Игра - это концентрация на процессе, игровое приложение — на результате. Игровое приложение строится на внешней мотивации: дети играют, чтобы получить виртуальные бонусы. Видеоигра строится на внутренней мотивации: дети играют не только ради наград, но и ради самой игры.

**5 Проектирование задачи**

**5.1 Разработка логической и физической структуры программы**

**5.1.1 Структура программы**

Данный проект содержит двенадцати классов. Далее рассмотрим назначение каждого класса:

Class AddRoom – класс, отвечающий за наполнение комнат, создания в них врагов и открытие дверей;

Class Bullet – класс, ответственный за поведение “пуль”;

Class ChangeRoom – класс, ответственный за переход персонажа и камеры между комнатами;

Class DeadEnd – класс, ответственный за экран смерти игрока;

Class Enemy – класс, для работы с врагами;

Class Gun – класс, в котором реализуется поведение оружия;

Class MainMenu – класс, ответственный за главное меню;

Class PauseMenu – класс, ответственный за меню паузы;

Class Player – класс, для работы с игроком;

Class RoomSpawner – класс, ответственный за процедурную генерацию;

Class RoomVariants – класс, для хранения вариантов комнат, создание комнаты с боссом и добавление ключа от дверей;

Class Wall – класс, для работы со стенами;

**5.1.2 Определение атрибутов классов**

Описание разработанных атрибутов находится в таблице 1.

Таблица 1 – Атрибуты

|  |  |  |
| --- | --- | --- |
| Имя атрибута | В каком классе | Назначение |
| 1 | 2 | 3 |
| walls | AddRoom | Массив со всеми выходами из комнаты. |
| door | AddRoom | Дверь с замком. |
| portal | AddRoom | Портал. |
| enemyTypes | AddRoom | Массив с типами врагов. |
| enemySpawner | AddRoom | Массив с точками появления врагов. |
| enemies | AddRoom | Список врагов в комнате. |
| isBossRoom | AddRoom | Определяет, является ли комната комнатой с боссом. |
| isBossDefeated | AddRoom | Определяет, побежден ли босс. |
| isLastRooom | AddRoom | Определяет, является ли комната последней. |
| variants | AddRoom | Массив созданных комнат. |
| spawned | AddRoom | Определяет, добавлены ли враги в комнаты. |
| wallsDestroyed | AddRoom | Определяет, открыты ли проходы. |
| speed | Bullet | Скорость пули. |
| lifetime | Bullet | Время существования пули. |
| distance | Bullet | Место попадания пули. |
| damage | Bullet | Урон пули. |
| whatIsSolid | Bullet | Определяет, является ли объект твердым. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| destroyEffect | Bullet | Эффект столкновения пули. |
| cameraChangePosition | Bullet | Расстояние на которое камера изменит свою позицию. |
| cameraChangePosition | ChangeRoom | Расстояние на которое игрок изменит свою позицию. |
| cam | ChangeRoom | Количество кадров. |
| stopTime | Enemy | Время остановки врага. |
| startStopTime | Enemy | Начальное время остановки врага. |
| anim | Enemy | Аниматор. |
| player | Enemy | Игрок. |
| room | Enemy | Комната. |
| destroyEffect | Enemy | Эффект атаки. |
| timeBtwAttack | Enemy | Время между атаками. |
| starTimeBtwAttack | Enemy | Начальное время между атаками. |
| speed | Enemy | Скорость врага. |
| health | Enemy | Количество здоровья врага. |
| damage | Enemy | Урон врага. |
| playerNotInRoom | Enemy | Определяет, находится ли игрок в комнате. |
| isBoss | Enemy | Определяет, является ли враг боссом. |
| stopped | Enemy | Определяет остановился ли враг. |
| bullet | Gun | Пуля. |
| shotPoint | Gun | Точка вестрела. |
| gunType | Gun | Тип оружия. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| startTimeBtwShots | Gun | Начальное время между выстрелами. |
| timeBtwShots | Gun | Время между выстрелами. |
| offset | Gun | Отклонение направления оружия от мыши. |
| rotZ | Gun | Угол поворота. |
| difference | Gun | Разница между позицией мыши и положением оружия. |
| player | Gun | Игрок. |
| isPaused | PauseMenu | Определяет, стоит ли игра на паузе. |
| pauseMenuUI | PauseMenu | Меню паузы. |
| speed | Player | Скорость игрока. |
| health | Player | Количество здоровья игрока. |
| healthDisplay | Player | Счетчик здоровья на интерфейсе игрока. |
| level | Player | Счетчик уровня на интерфейсе игрока. |
| floor | Player | Счетчик этажа на интерфейсе игрока. |
| rb | Player | Твердое тело. |
| moveInput | Player | Управление передвижением. |
| moveVelocity | Player | Скорость передвижения. |
| anim | Player | Аниматор. |
| facingRight | Player | Определяет, направление игрока. |
| keyButtonPushed | Player | Определяет, нажата ли кнопка ключа. |
| levelNum | Player | Счетчик пройденных уровней. |
| floorNum | Player | Счетчик пройденных этажей. |
| mainRoom | Player | Главная комната. |
| Room | Player | Наполнитель комнат. |
| deadScreen | Player | Экран смерти игрока. |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| HUD | Player | Визуальный интерфейс игрока. |
| direction | RoomSpawner | Расположение точки добавления относительно комнаты. |
| variants | RoomSpawner | Варианты комнат. |
| rand | RoomSpawner | Случайное число. |
| spawned | RoomSpawner | Определяет, добавлена ли комната. |
| waitTime | RoomSpawner | Время ожидания. |
| topRooms | RoomVariants | Массив комнат, у которых есть проход вверх. |
| bottomRooms | RoomVariants | Массив комнат, у которых есть проход вниз. |
| leftRooms | RoomVariants | Массив комнат, у которых есть проход влево. |
| rightRooms | RoomVariants | Массив комнат, у которых есть проход вправо. |
| key | RoomVariants | Ключ |
| player | RoomVariants | Игрок |
| rooms | RoomVariants | Список добавленных комнат |
| block | Wall | Блок из которого состоят комнаты. |

**3.3 Определение методов класса**

Описание использованных методов класса приводиться в таблице 2.

Таблица 2 – Использованные методы класса

|  |  |  |
| --- | --- | --- |
| Метод | Класс | Назначение |
| 1 | 2 | 3 |
| private void Awake() | AddRoom | Метод для получения списка добавленных комнат. |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| private void Start() | AddRoom | Метод для добавления в список комнат текущей комнате. |
| private void OnTriggerEnter2D(Collider2D other) | AddRoom | Метод для добавления врагов в комнаты. |
| IEnumerator CheckEnemies() | AddRoom | Метод для подсчета врагов и активации портала. |
| public void DestroyWalls() | AddRoom | Метод для открытия проходов. |
| private void OnTriggerStay2D(Collider2D other) | AddRoom | Метод для открытия прохода в соседней комнате. |
| private void OnTriggerExit2D(Collider2D other) | AddRoom | Метод для остановки врагов если игрок находится вне комнаты. |
| private void Start() | Bullet | Метод для добавления пули. |
| void Update() | Bullet | Метод для установки поведения пули. |
| public void DestroyBullet() | Bullet | Метод для уничтожения пули. |
| private void Start() | ChangeRoom | Метод для определения главной камеры. |
| private void OnTriggerEnter2D(Collider2D other) | ChangeRoom | Метод для изменения позиции игрока и камеры. |
| private void Update() | DeadEnd | Метод для остановки времени в игре. |
| public void LoadMenu() | DeadEnd | Метод для перехода в главное меню. |
| void Start() | Enemy | Метод для установки значений атрибутов класса. |
| void Update() | Enemy | Метод для установки поведения врагов. |
| public void TakeDamage(int damage) | Enemy | Метод для получения врагами урона |
| private void OnTriggerStay2D(Collider2D other) | Enemy | Метод для атаки игрока врагами. |
| public void OnEnemyAttack() | Enemy | Метод для установки результата атаки игрока врагами. |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| void Start() | Gun | Метод для установки значений атрибутов класса. |
| void Update() | Gun | Метод для установки поведения оружия. |
| public void Shoot() | Gun | Метод для выстрела из оружия. |
| public void PlayGame() | MainMenu | Метод для начала игры. |
| public void ExitGame() | MainMenu | Метод для выхода из игры. |
| void Update() | PauseMenu | Метод для включения или выключения паузы по кнопке Escape. |
| public void Resume() | PauseMenu | Метод для выключения паузы. |
| public void LoadMenu() | PauseMenu | Метод для перехода в главное меню. |
| void Pause() | PauseMenu | Метод для включения паузы. |
| void Start() | Player | Метод для установки значений атрибутов класса. |
| void Update() | Player | Метод для управления игроком. |
| private void FixedUpdate() | Player | Метод для коррекции передвижения. |
| private void Flip() | Player | Метод для поворота игрока. |
| public void ChangeHealth(int healthValue) | Player | Метод для изменения здоровья игрока. |
| private void OnTriggerEnter2D(Collider2D other) | Player | Метод для включения кнопки ключа. |
| public void onKeyButtonDown() | Player | Метод для проверки нажатия кнопки ключа. |
| private void OnTriggerStay2D(Collider2D other) | Player | Метод для открытия дверей с замком и перехода на новый этаж или уровень. |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| private void Start() | RoomSpawner | Метод для установки значений атрибутов класса. |
| public void Spawn() | RoomSpawner | Метод для добавления комнат. |
| private void OnTriggerStay2D(Collider2D other) | RoomSpawner | Метод для удаления точки добавления. |
| private void Start() | RoomVariants | Метод для установки значений атрибутов класса. |
| IEnumerator RandomSpawner() | RoomVariants | Метод добавления ключа и дверей с замком в случайные комнаты. |
| private void OnTriggerEnter2D(Collider2D other) | Wall | Метод для замены прохода на стену. |
| private void OnTriggerStay2D(Collider2D other) | Wall |  |

**5.2 Процессы**

Выделим основные процессы, из которых будет состоять разрабатываемое игровое приложение. Их полное описание приведено в таблице 1.

Таблица 3 – Процессы

|  |  |
| --- | --- |
| Процесс | Описание процесса |
| 1 | 2 |
| 1 Начать игру | 1.1 Передвижение персонажа |
| 1.2 Нажатие на кнопки |
| 1.3 Стрельба |

**6** **Реализация**

* 1. **Руководство программиста**

**6.1.1 Структура программы**

Программа представляет собой последовательность уровней, которые необходимо пройти пользователю.

* + 1. **Структура и описание процедур и функций пользователя**

Пользователь может управлять авакадо, который необходимо довести в конец уровня параллельно сражаясь с врагами. Помимо этого он может открывать проходы запертые на замок.

* + 1. **Спецификация программы**

Данное игровое приложение поможет пользователю хорошо провести свободное время, поскольку в наше время много кто любит играть в компьютерные игры.

* + 1. **Структура со стороны кода**

Unity это уникальная среда разработки. Во время создания приложения практически не возникало ошибок. Также в этой среде разработки было удобно писать код и при обнаружении ошибки, быстро их устранять

**7 Тестирование**

Тестирование является одним из важных этапов при создании и внедрении программного продукта. Исходя из поведения программы при определенном наборе тестов, нельзя сделать однозначных выводов о том, что программа будет исправно работать в любой момент использования. Опираясь на данный набор тестов, можно говорить о некоторой степени уверенности в правильности работы программного продукта.

Тестирование программного продукта не проводилось, так как программный продукт находится на ранней стадии разработки.

**8 Применение**

**8.1 Назначение и область применения программы**

**8.1.1 Область применения**

Данный программный продукт предназначен для пользователей, которые хотят хорошо провести своё свободное время.

Данный курсовой проект прост и удобен в применении, для работы с ним не требуется высокого навыка пользование компьютером.

**8.1.2** **Наименование**

Игровое приложение “OnionFree”

**8.1.3 Программное обеспечение, необходимое для функционирования программы**

Функционирование программного продукта тестировалось в следующей операционной системе (ОС): Microsoft Windows 10.

**8.1.4 Языки программирования, на которых написана программа**

Игровое приложение «OnionFree» было реализовано на языке программирования высокого уровня C#.

Программа предназначена, чтобы в неё играть.

Условия, необходимые выполнения программы:

- процессор Intel i3;

- объем ОЗУ 1 ГБ;

- видеосистема: 1024\*768 точек с глубиной цвета 16bit;

- операционная система Windows XP.

**8.2 Инсталляция**

Для того, чтобы установить программу, необходимо запустить файл Setup.exe. Появится окно игрового-развлекательного приложения OnionFree (рисунок 3).

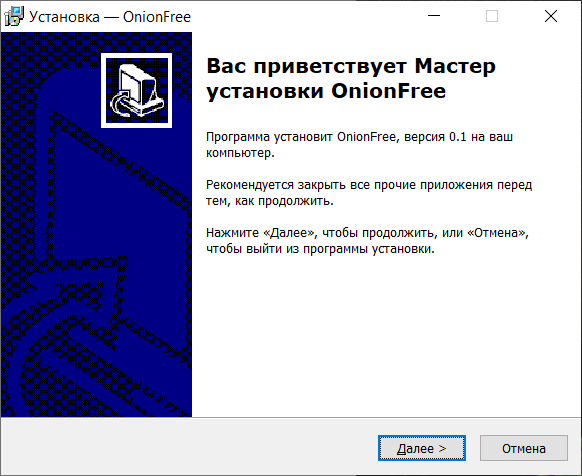


Рисунок 3 – Установка игрового приложения OnionFree

После нажатия на кнопку «Далее» появляется окно выбора каталога установки (рисунок 4). Нажав на кнопку «обзор», вы можете поменять место для сохранения файла.

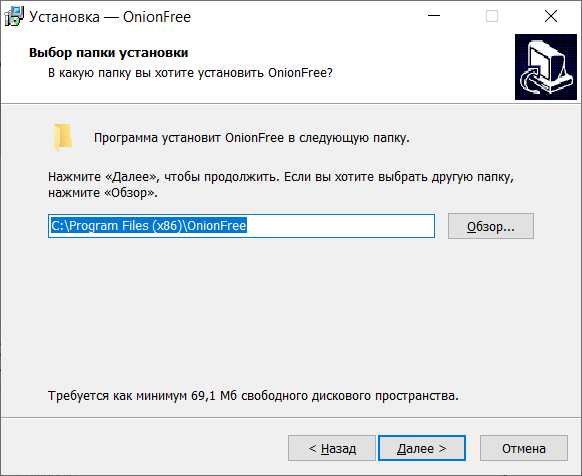


Рисунок 4 – выбор каталога установки

Выбрав все необходимое для установки программного продукта, вам будет предложено нажать на кнопку «установить». После её нажатия пойдет загрузка файлов и приложение будет готово к работе.

**8.3 Выполнения программы**

**8.3.1 Запуск программы**

Данную программу можно запустить различными способами.

Первым из них является запуск с помощью ярлыка на рабочем столе. Необходимо дважды кликнуть левой кнопкой мышки на ярлыке (рисунок 5).



Рисунок 5 – Ярлык приложения OnionFree

Вторым способом является запуск поиск по названию. Необходимо ввести “OnionFree”.

Третьим способ является запуск из каталога, в которое устанавливалось приложение (По умолчанию C:\Program Files (x86)\OnionFree).

**8.3.2 Инструкция по работе с программой**

После запуска программы на экране появляется заставка (рисунок 6).



Рисунок 6 – заставка приложения

После заставки появляется главная меню (рисунок 7) откуда пользователь может перейти в любую часть самого приложения.

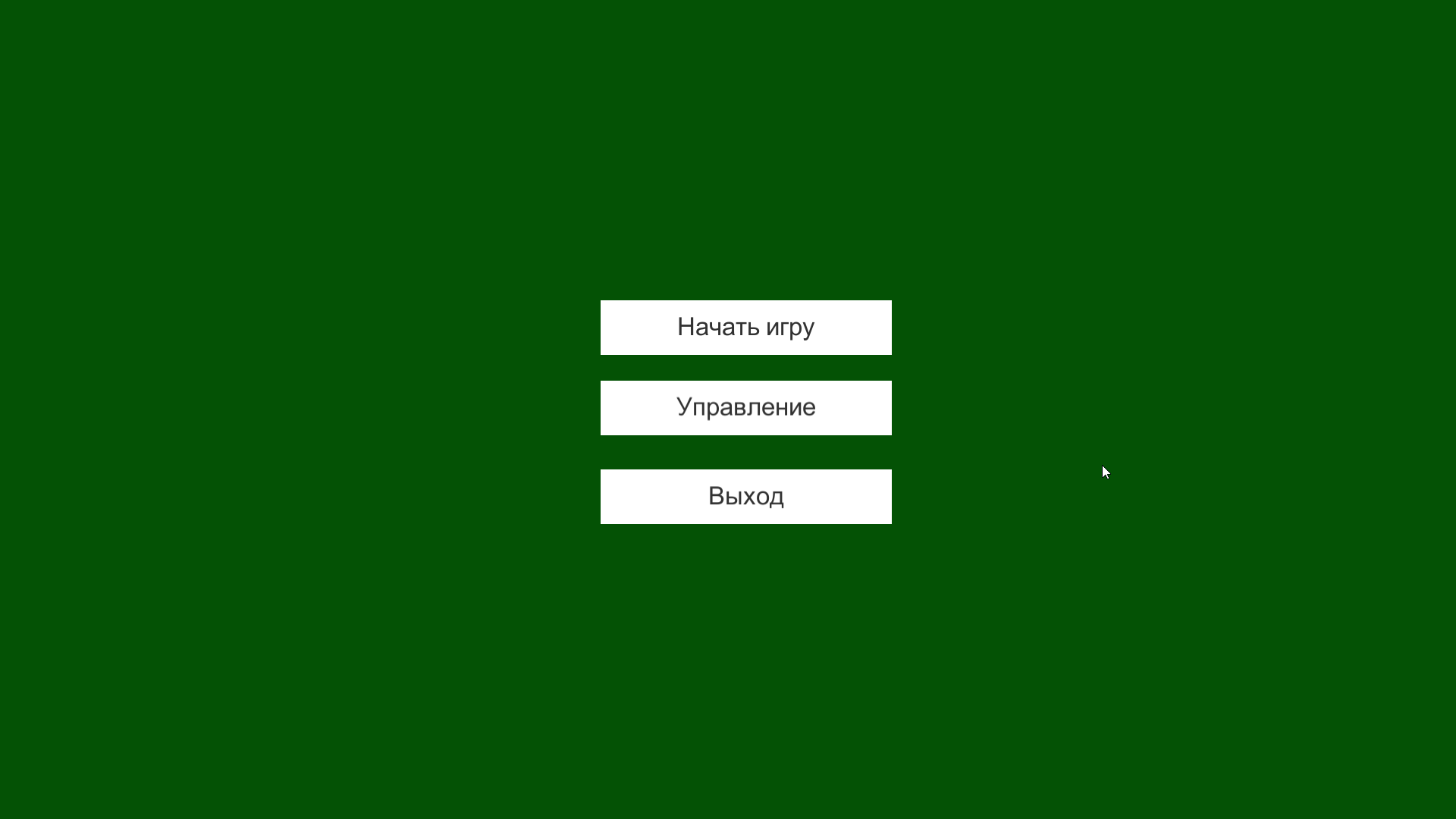


Рисунок 7 – главное меню

Далее у пользователя появляется возможность выбора того, что он хочет делать в этом приложении.

При нажатии на кнопку «Начать игру» пользователя на сцену с игрой (рисунок 8).



Рисунок 8 – Игра

Для открытия информации об управлении нужно нажать на кнопку «Управление» (рисунок 9).



Рисунок 9 – Управление

**8.3.3 Завершение работы программы**

Для выхода из приложения нужно нажать «Выход» в главном меню (рисунок 10).



Рисунок 10 – Выход

**Заключение**

Целью данного проекта было разработать игровое приложение «OnionFree».

В поставленной задаче был реализован простой, понятный, удобный, интересный, красивый пользовательский интерфейс, который соответствует тематике игре.

В ходе тестирования все исключительные ситуации были обработаны. Проект работает без сбоев и ошибок.

Продукт имеет дальнейшее развитие. Можно выпустить обновление, добавить новых врагов уровни и так далее. Но несмотря на все это, данный программный продукт имеет законченный вид.

Исходя из этого, можно сделать вывод, что программа реализована успешно, однако она требует дальнейших изменений.

**Список использованных источников**

1. Плейлист Top Down Шутер / Roguelike || Unity C# Уроки Создание Игр 2D – YouTube: <https://youtube.com/playlist?list=PLRHtm1zQx-f_vzWJ7oOXOqyWMhbLB2QK9>
2. 2D шутер || Unity Уроки C# 2D – YouTube: <https://youtube.com/playlist?list=PLRHtm1zQx-f8bHf4o8dIbQ0_zvOoTZ1Tb>
3. Бесконечный 2D раннер || Unity Уроки C# - YouTube: <https://youtube.com/playlist?list=PLRHtm1zQx-f-6WDUqRsyzVJ-BLWYTZN7M>
4. АНИМАЦИЯ ПЕРСОНАЖА В UNITY #2 || Unity C# Уроки Создание Игр На Андроид 2D: <https://youtu.be/mpdthsdB-W0>

**­­Приложение А**

**Листинг программы**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AddRoom : MonoBehaviour

{

[Header("Walls")]

public GameObject[] walls;

public GameObject door;

[Header("Walls")]

public GameObject portal;

[Header("Enemies")]

public GameObject[] enemyTypes;

public GameObject healthPotion;

public Transform[] enemySpawner;

[HideInInspector] public List<GameObject> enemies;

[HideInInspector] public bool isBossRoom;

[HideInInspector] public bool isBossDefeated;

[HideInInspector] public bool isLastRoom;

private RoomVariants variants;

private bool spawned;

private bool wallsDestroyed;

public RoomVariants RoomVariants

{

get => default;

set

{

}

}

public Player Player

{

get => default;

set

{

}

}

public Enemy Enemy

{

get => default;

set

{

}

}

private void Awake()

{

variants = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomVariants>();

}

private void Start()

{

variants.rooms.Add(gameObject);

}

private void OnTriggerEnter2D(Collider2D other)

{

if(other.CompareTag("Player") && !spawned)

{

spawned = true;

if (!isBossRoom && gameObject.name != "MainRoom" && gameObject.name != "MainRoom(Clone)")

{

foreach (Transform spawner in enemySpawner)

{

int rand = Random.Range(0, 10);//до 11

if (rand < 9)

{

GameObject enemyType = enemyTypes[Random.Range(0, enemyTypes.Length)];

GameObject enemy = Instantiate(enemyType, spawner.position, Quaternion.identity) as GameObject;

enemy.transform.SetParent(gameObject.transform);

enemies.Add(enemy);

}

else if (rand == 9)

{

Instantiate(healthPotion, spawner.position, Quaternion.identity);

}

/\*else if (rand == 10)

{

Instantiate(shield, spawner.position, Quaternion.identity);

}\*/

}

}

else if (isBossRoom && gameObject.name != "MainRoom" && gameObject.name != "MainRoom(Clone)")

{

int rand = Random.Range(0, enemySpawner.Length);

GameObject enemy = Instantiate(enemyTypes[0], enemySpawner[rand].position, Quaternion.identity) as GameObject;

enemy.GetComponent<Enemy>().isBoss = true;

enemy.GetComponent<Enemy>().health \*= 2;

enemy.GetComponent<Enemy>().speed = 1;

enemy.GetComponent<Enemy>().damage = 2;

enemy.GetComponent<Enemy>().isBoss = true;

enemy.transform.localScale = new Vector3((float)(gameObject.transform.localScale.x \* 0.5), (float)(gameObject.transform.localScale.y \* 0.5));

enemy.transform.SetParent(gameObject.transform);

enemies.Add(enemy);

}

else

{

DestroyWalls();

}

StartCoroutine(CheckEnemies());

} else if (other.CompareTag("Player") && spawned)

{

foreach(GameObject enemy in enemies)

{

enemy.GetComponent<Enemy>().playerNotInRoom = false;

}

}

}

IEnumerator CheckEnemies()

{

yield return new WaitForSeconds(1f);

yield return new WaitUntil(() => enemies.Count == 0);

if (isBossRoom && isBossDefeated || isLastRoom)

portal.SetActive(true);

DestroyWalls();

}

public void DestroyWalls()

{

foreach(GameObject wall in walls)

{

if(wall != null && wall.transform.childCount != 0)

{

Destroy(wall);

}

}

wallsDestroyed = true;

}

private void OnTriggerStay2D(Collider2D other)

{

if(wallsDestroyed && other.CompareTag("Wall"))

{

Destroy(other.gameObject);

}

}

private void OnTriggerExit2D(Collider2D other)

{

if (other.CompareTag("Player") && spawned)

{

foreach (GameObject enemy in enemies)

{

enemy.GetComponent<Enemy>().playerNotInRoom = true;

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Bullet : MonoBehaviour

{

public float speed;

public float lifetime;

public float distance;

public int damage;

public LayerMask whatIsSolid;

public GameObject destroyEffect;

[SerializeField] bool enemyBullet;

public Gun Gun

{

get => default;

set

{

}

}

public Enemy Enemy

{

get => default;

set

{

}

}

private void Start()

{

Invoke("DestroyBullet", lifetime);

}

// Update is called once per frame

void Update()

{

RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, transform.up, distance, whatIsSolid);

if(hitInfo.collider != null)

{

if(hitInfo.collider.CompareTag("Enemy"))

hitInfo.collider.GetComponent<Enemy>().TakeDamage(damage);

if (hitInfo.collider.CompareTag("Player") && enemyBullet)

hitInfo.collider.GetComponent<Player>().ChangeHealth(-damage);

DestroyBullet();

}

transform.Translate(Vector2.up \* speed \* Time.deltaTime);

}

public void DestroyBullet()

{

Instantiate(destroyEffect, transform.position, Quaternion.identity);

Destroy(gameObject);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ChangeRoom : MonoBehaviour

{

public Vector3 cameraChangePos;

public Vector3 playerChangePos;

private Camera cam;

public Player Player

{

get => default;

set

{

}

}

private void Start()

{

cam = Camera.main.GetComponent<Camera>();

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Player"))

{

other.transform.position += playerChangePos;

cam.transform.position += cameraChangePos;

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class DeadEnd : MonoBehaviour

{

public MainMenu MainMenu

{

get => default;

set

{

}

}

private void Update()

{

Time.timeScale = 0f;

}

public void LoadMenu()

{

Time.timeScale = 1f;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

public void Restart()

{

Time.timeScale = 1f;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Enemy : MonoBehaviour

{

private float stopTime;

public float startStopTime;

private Animator anim;

private Player player;

private AddRoom room;

public GameObject destroyEffect;

private float timeBtwAttack;

public float startTimeBtwAttack;

public float speed;

public int health;

public int damage;

[HideInInspector] public bool playerNotInRoom;

[HideInInspector] public bool isBoss;

private bool stopped;

public Player Player

{

get => default;

set

{

}

}

public AddRoom AddRoom

{

get => default;

set

{

}

}

// Start is called before the first frame update

void Start()

{

anim = GetComponent<Animator>();

player = FindObjectOfType<Player>();

room = GetComponentInParent<AddRoom>();

stopped = false;

}

// Update is called once per frame

void Update()

{

if (!playerNotInRoom)

{

if (stopTime <= 0)

stopped = false;

else

{

stopped = true;

stopTime -= Time.deltaTime;

}

}

else

{

stopped = true;

}

if (room == null)

{

Debug.Log("Room is Null");//Попробуй присваивать комнату волкам при спавне

}

if (health <= 0)

{

room.enemies.Remove(gameObject);

if (isBoss)

room.isBossDefeated = true;

Destroy(gameObject);

}

if (player.transform.position.x > transform.position.x)

transform.eulerAngles = new Vector3(0, 180, 0);

else

transform.eulerAngles = new Vector3(0, 0, 0);

if(!stopped)

transform.position = Vector2.MoveTowards(transform.position, player.transform.position, speed \* Time.deltaTime);

}

public void TakeDamage(int damage)

{

stopTime = startStopTime;

health -= damage;

}

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Player"))

{

if (timeBtwAttack <= 0)

{

anim.SetTrigger("attack");

}

else

{

timeBtwAttack -= Time.deltaTime;

}

}

}

public void OnEnemyAttack()

{

Instantiate(destroyEffect, player.transform.position, Quaternion.identity);

player.ChangeHealth(-damage);

timeBtwAttack = startTimeBtwAttack;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Gun : MonoBehaviour

{

public GameObject bullet;

public Transform shotPoint;

public GunType gunType;

public float startTimeBtwShots;

public float offset;

public enum GunType { Default, Enemy }

private float timeBtwShots;

private float rotZ;

private Vector3 difference;

private Player player;

// Start is called before the first frame update

void Start()

{

player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

}

// Update is called once per frame

void Update()

{

if (gunType == GunType.Default)

{

difference = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;

rotZ = Mathf.Atan2(difference.y, difference.x) \* Mathf.Rad2Deg;

}

else if (gunType == GunType.Enemy)

{

difference = player.transform.position - transform.position;

rotZ = Mathf.Atan2(difference.y, difference.x) \* Mathf.Rad2Deg;

}

transform.rotation = Quaternion.Euler(0f, 0f, rotZ + offset);

if (timeBtwShots <= 0)

{

if (Input.GetMouseButton(0) || gunType == GunType.Enemy)

Shoot();

} else

{

timeBtwShots -= Time.deltaTime;

}

}

public void Shoot()

{

Instantiate(bullet, shotPoint.position, shotPoint.rotation);

timeBtwShots = startTimeBtwShots;

}

public Player Player

{

get => default;

set

{

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine;

public class MainMenu : MonoBehaviour

{

public RoomSpawner RoomSpawner

{

get => default;

set

{

}

}

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

public void ExitGame()

{

Debug.Log("Exit");

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour

{

public static bool isPaused = false;

public GameObject pauseMenuUI;

public MainMenu MainMenu

{

get => default;

set

{

}

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyUp(KeyCode.Escape))

if (isPaused)

Resume();

else

Pause();

}

public void Resume()

{

pauseMenuUI.SetActive(false);

Time.timeScale = 1f;

isPaused = false;

}

public void Restart()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

public void LoadMenu()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

void Pause()

{

pauseMenuUI.SetActive(true);

Time.timeScale = 0f;

isPaused = true;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class Player : MonoBehaviour

{

public float speed;

public int health;

public int maxHealth;

public Text healthDisplay;

public Text level;

public Text floor;

private Rigidbody2D rb;

private Vector2 moveInput;

private Vector2 moveVelocity;

private Animator anim;

private bool facingRight = true;

private bool keyButtonPushed;

private int levelNum;

private int floorNum;

public GameObject mainRoom;

public GameObject Room;

public GameObject deadScreen;

public GameObject HUD;

[Header("Key")]

public GameObject keyIcon;

public RoomVariants RoomVariants

{

get => default;

set

{

}

}

public PauseMenu PauseMenu

{

get => default;

set

{

}

}

public DeadEnd DeadEnd

{

get => default;

set

{

}

}

/\*public GameObject portalIcon;\*/

// Start is called before the first frame update

void Start()

{

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

levelNum = int.Parse(level.text);

floorNum = int.Parse(floor.text);

Time.timeScale = 1f;

}

// Update is called once per frame

void Update()

{

healthDisplay.text = health.ToString();

moveInput = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));

moveVelocity = moveInput.normalized \* speed;

if(moveInput.x == 0)

anim.SetBool("isRunning", false);

else

anim.SetBool("isRunning", true);

if (!facingRight && moveInput.x>0)

Flip();

else if(facingRight && moveInput.x<0)

Flip();

if (health <= 0)

{

deadScreen.SetActive(true);

HUD.SetActive(false);

}

}

private void FixedUpdate()

{

rb.MovePosition(rb.position + moveVelocity \* Time.fixedDeltaTime);

}

private void Flip()

{

facingRight = !facingRight;

Vector3 Scaler = transform.localScale;

Scaler.x \*= -1;

transform.localScale = Scaler;

}

public void ChangeHealth(int healthValue)

{

health += healthValue;

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Key"))

{

keyIcon.SetActive(true);

Destroy(other.gameObject);

}

if (other.CompareTag("Heal"))

{

if (health < maxHealth)

{

ChangeHealth(3);

Destroy(other.gameObject);

}

}

}

public void onKeyButtonDown()

{

keyButtonPushed = !keyButtonPushed;

}

/\*public void onPortalBurronDown()

{

portalButtonPushed = !portalButtonPushed;

}\*/

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Door") && keyButtonPushed && keyIcon.activeInHierarchy)

{

keyIcon.SetActive(false);

other.gameObject.SetActive(false);

keyButtonPushed = false;

}

if (other.CompareTag("Portal"))

{

if (int.Parse(floor.text) < 3)

{

floorNum++;

floor.text = floorNum.ToString();

}

else

{

levelNum++;

level.text = levelNum.ToString();

floorNum = 1;

floor.text = floorNum.ToString();

}

if (level.text != "3")

{

GameObject[] rooms = GameObject.FindGameObjectsWithTag("Room");

foreach(var room in rooms)

{

Destroy(room);

}

Instantiate(Room, Camera.main.GetComponent<Camera>().transform.position, Quaternion.identity);

Instantiate(mainRoom, Camera.main.GetComponent<Camera>().transform.position, Quaternion.identity);

gameObject.transform.position = Camera.main.GetComponent<Camera>().transform.position;

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomSpawner : MonoBehaviour

{

public Direction direction;

public enum Direction

{

Top,

Bottom,

Left,

Right,

None

}

private RoomVariants variants;

private int rand;

private bool spawned = false;

private float waitTime = 3f;

private void Start()

{

variants = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomVariants>();

Invoke("Spawn", 0.2f);

Destroy(gameObject, waitTime);

}

public void Spawn()

{

if (!spawned)

{

if(direction == Direction.Top)

{

rand = Random.Range(0, variants.topRooms.Length);

Instantiate(variants.topRooms[rand], transform.position, variants.topRooms[rand].transform.rotation);

} else if (direction == Direction.Bottom)

{

rand = Random.Range(0, variants.bottomRooms.Length);

Instantiate(variants.bottomRooms[rand], transform.position, variants.bottomRooms[rand].transform.rotation);

}else if (direction == Direction.Left)

{

rand = Random.Range(0, variants.leftRooms.Length);

Instantiate(variants.leftRooms[rand], transform.position, variants.leftRooms[rand].transform.rotation);

}else if (direction == Direction.Right)

{

rand = Random.Range(0, variants.rightsRooms.Length);

Instantiate(variants.rightsRooms[rand], transform.position, variants.rightsRooms[rand].transform.rotation);

}

spawned = true;

}

}

private void OnTriggerStay2D(Collider2D other)

{

if(other.CompareTag("RoomPoint") && other.GetComponent<RoomSpawner>().spawned)

{

Destroy(gameObject);

}

}

public AddRoom AddRoom

{

get => default;

set

{

}

}

public RoomVariants RoomVariants

{

get => default;

set

{

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomVariants : MonoBehaviour

{

public GameObject[] topRooms;

public GameObject[] bottomRooms;

public GameObject[] leftRooms;

public GameObject[] rightsRooms;

public GameObject key;

private Player player;

[HideInInspector] public List<GameObject> rooms;

private void Start()

{

StartCoroutine(RandomSpawner());

player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

}

IEnumerator RandomSpawner()

{

yield return new WaitForSeconds(5f);

AddRoom lastRoom = rooms[rooms.Count - 1].GetComponent<AddRoom>();

lastRoom.isLastRoom = true;

if (int.Parse(player.floor.text) == 3)

lastRoom.isBossRoom = true;

int rand = Random.Range(0, rooms.Count - 2);

Instantiate(key, rooms[rand].transform.position, Quaternion.identity);

lastRoom.door.SetActive(true);

lastRoom.DestroyWalls();

Destroy(gameObject);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Wall : MonoBehaviour

{

public GameObject block;

public Wall Wall1

{

get => default;

set

{

}

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Block"))

{

Debug.Log("Block behind the Wall!");

Instantiate(block, transform.GetChild(0).position, Quaternion.identity);

Instantiate(block, transform.GetChild(1).position, Quaternion.identity);

Destroy(gameObject);

}

}

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Block"))

{

Debug.Log("Block behind the Wall!");

Instantiate(block, transform.GetChild(0).position, Quaternion.identity);

Instantiate(block, transform.GetChild(1).position, Quaternion.identity);

Destroy(gameObject);

}

}

}