**Введение**

На учебную практику была поставлена задача разработки игрового приложения в жанре 2D-платформер «ShaDDow».

Цель учебной практики заключается в разработке программного продукта, для использования в свободное время.

Создаваемая программа будет рассчитана на любых пользователей. Программой сможет воспользоваться любой желающий любого возраста.

Далее приведем краткое описание разделов пояснительной записки.

Первый раздел носит название “Анализ задачи”. В нем вы можете ознакомиться с постановкой задачи, которая включает в себя: исследование предметной области поставленной задачи. Также в этом разделе вы можете узнать о том, как данная задача решается в настоящее время. Все входные и выходные данные тоже описаны в первом разделе. В подразделе “Инструменты разработки” рассматривается среда, в которой создается данный курсовой проект. Здесь также установлены минимальные и оптимальные требования к аппаратным характеристикам, обеспечивающим правильное функционирование поставленной задачей.

В разделе “Проектирование задачи” рассмотрены основные аспекты разработки программного продукта. Здесь можно узнать об организации данных в контексте среды разработки. В данном разделе четко описан пользовательский интерфейс, составлены алгоритмы процесса обработки информации, описана разработка системы справочной информации.

“Реализация задачи” – это третий раздел пояснительной записки, в котором описываются все элементы и объекты, которые будут использованы при реализации данного приложения. В этом разделе четко описаны функции пользователя и их структура. Здесь можно найти таблицу, в которой будет представлена полная аннотация файлов, используемых в данном проекте.

Четвертый раздел – “Тестирование”. В нем описано полное и функциональное тестирование данной программы, т.е. оттестирован каждый пункт меню, каждая операция, которая выполняется приложением. Смоделированы все возможные действия пользователя при работе с программой, начиная от запуска до выхода.

В разделе “Применение” описано назначение, область применения, среда функционирования курсовой программы.

“Заключение” содержит краткую формулировку задачи, результаты проделанной работы, описание использованных методов и средств, описание степени автоматизации процессов на различных этапах разработки.

В “Литературе” приведен список используемых при разработке источников.

В приложениях к пояснительной записке приведен листинг программы с необходимыми комментариями.

Схема работы системы представлена в графической части.

**1 Анализ задачи**

**1.1 Постановка задачи**

**1.1.1 Организационно-экономическая сущность задачи**

Темой данного проекта является приложение «ShaDDow».

Это простая бесплатная 2D-игра для веселого провождения времени. В ней вы играете за маленькую тень в темном лесу которой предстоит пройти полосу препятствий всячески избегая ловушек и монстров. Но для любителей PVP в игре реализована функция атаки, и если вам не хочется избегать врагов уворачиваясь от атак, то вы вполне можете пойти на пролом, но помните, что жизни не бесконечны.

Цель проекта: предоставить пользователям приложение в котором они смогут развить реакцию, проявить логику и смикалку.

Периодичность использования данного программного продукта не ограничена. Пользователь может в любое время установить, а также удалить приложение.

Аналоги такого приложения можно найти на разнообразных веб-ресурсах (Сайтах, форумах, социальных сетях, игровых площадках). Жанр достаточно популярный и креативных проектов очень и очень мало.

**1.1.2 Функциональные требования**

Разрабатываемый программный продукт позволит выполнить следующие действия

– запуск игры

– переход на другую локацию

– выход из игры;

**1.1.3 Эксплуатационные требования**

Требования к реализации: Для полной реализации будет использоваться язык C#, с игровым движком Unity Engine.

Требования к надежности: alfa- и beta-тестирования перед выпуском новых обновлений.

Требования к интерфейсу: интерфейс должен быть соответсвовать пиксельной графике) с приятной цветовой гаммой (в оттенках синего цвета) и понятной для пользователя. Следовательно, каждое окно должно иметь ясную визуальную иерархию своих элементов. Фрагменты текста должны располагаться на экране так, чтобы пользователя было просто и понятно принимать информацию.

**1.2 Диаграмма вариантов использования**

Диаграмма вариантов использования – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования представлена на рисунке 1

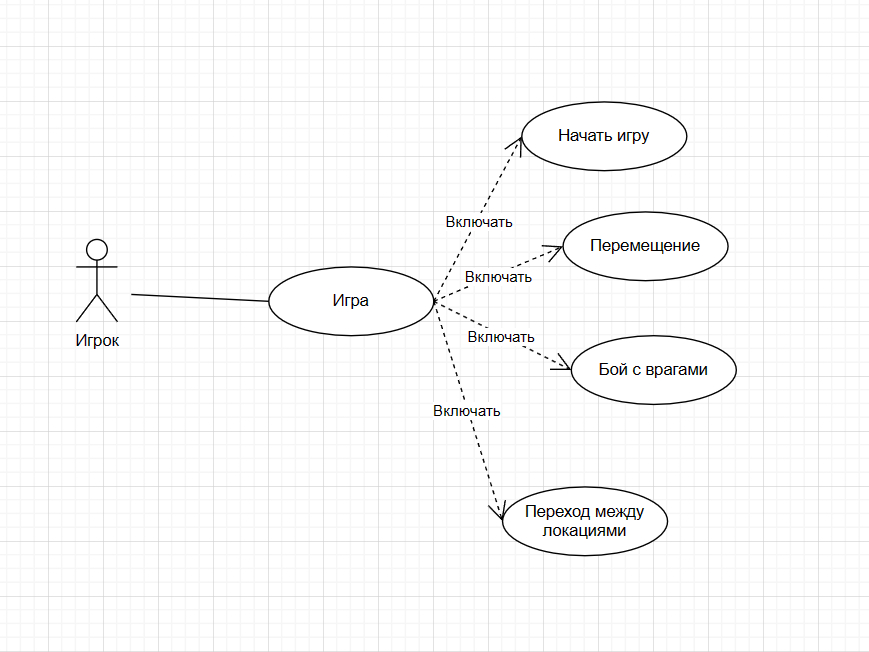


Рисунок 1 - Диаграмма вариантов использования для пользователя

**1.3 Выбор стратегии разработки и модели жизненного**

Для разработки игры «Nomi» следует выбрать стратегию разработки и модель жизненного цикла. Осуществляем выбор посредством составления таблиц:

Таблица 1 – Выбор модели жизненного цикла на основе характеристик требований

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1. | Являются ли требования к проекту легко определимыми и реализуемыми? | Да | Да | Да | Нет | Нет | Нет |
| 2. | Могут ли требования быть сформулированы в начале ЖЦ? | Да | Да | Да | Да | Нет | Нет |
| 3. | Часто ли будут изменяться требования на протяжении ЖЦ? | Нет | Нет | Нет | Нет | Да | Да |
| 4. | Нужно ли демонстрировать требования с целью их определения? | Нет | Нет | Да | Нет | Да | Да |
| 5. | Требуется ли проверка концепции программного средства или системы? | Нет | Нет | Да | Нет | Да | Да |
| 6. | Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |

Вычисления: 2 за каскадную, за 2 V- образную, 5 за RAD, 3 за инкрементную, 5 за быстрого прототипирования и 5 за эволюционную.

Итог: На основе результатов заполнения табл. 3 подходящей является модель быстрого прототипирования и эволюционная.

Таблица 2 – Выбор модели жизненного цикла на основе характеристик команды разработчиков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории команды разработчиков  проекта | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1. | Являются ли проблемы предметной области проекта новыми для большинства разработчиков? | Нет | Нет | Нет | Нет | Да | Да |
| 2. | Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков? | Да | Да | Нет | Нет | Нет | Да |
| 3. | Изменяются ли роли участников проекта на протяжении ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Является ли структура процесса разработки более значимой для разработчиков, чем гибкость? | Да | Да | Нет | Да | Нет | Нет |
| 5. | Важна ли легкость распределения человеческих ресурсов проекта? | Да | Да | Да | Да | Нет | Нет |
| 6. | Приемлет ли команда разработчиков оценки, проверки, стадии разработки? | Да | Да | Нет | Да | Да | Да |

Вычисления: 4 за каскадную, 4 за V-образную, 1 за RAD, 4 за инкрементную, 3 за быстрого прототипирования и 4 за эволюционную.

Итог: На основе результатов заполнения табл. 4 подходящими являются каскадная, V-образная, инкрементная и эволюционная модели.

Таблица 3 – Выбор модели жизненного цикла на основе характеристик коллектива пользователей

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории коллектива пользователей | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Будет ли присутствие пользователей ограничено в ЖЦ разработки? | Да | Да | Нет | Да | Нет | Да |
| 2. | Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки? | Нет | Нет | Нет | Да | Да | Да |
| 3. | Будут ли пользователи вовлечены во все фазы ЖЦ разработки? | Нет | Нет | Да | Нет | Да | Нет |
| 4. | Будет ли заказчик отслеживать ход выполнения проекта? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 1 за каскадную, 1 за V-образную, 1 за RAD, 2 за инкрементную, 3 за быстрого прототипирования и 3 за эволюционную.

Итог: На основе результатов заполнения табл. 5 подходящей является модель быстрого прототипирования и эволюционная.

Таблица 6 **–** Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории типов проекта и рисков | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Разрабатывается ли в проекте продукт нового для организации направления? | Нет | Нет | Нет | Да | Да | Да |
| 2. | Будет ли проект являться расширением существующей системы? | Да | Да | Да | Да | Нет | Нет |
| 3. | Будет ли проект крупно- или среднемасштабным? | Нет | Нет | Нет | Да | Да | Да |

Продолжение таблицы 6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4. | Ожидается ли длительная эксплуатация продукта? | Да | Да | Нет | Да | Нет | Да |
| 5. | Необходим ли высокий уровень надежности продукта проекта? | Нет | Да | Нет | Да | Нет | Да |
| 6. | Предполагается ли эволюция продукта проекта в течение ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Велика ли вероятность изменения системы (продукта) на этапе сопровождения? | Нет | Нет | Нет | Да | Да | Да |
| 8. | Является ли график сжатым? | Нет | Нет | Да | Да | Да | Да |
| 9. | Предполагается ли повторное использование компонентов? | Нет | Нет | Да | Да | Да | Да |
| 10. | Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 3 за каскадную, 2 за V-образную, 6 за RAD, 6 за инкрементную, 7 за быстрого прототипирования и 6 за эволюционную.

Итог: На основе результатов заполнения табл. 6 подходящей являетсямодель быстрого прототипирования.

Общий итог: в итоге заполнения табл. 3 – 6 наиболее подходящей является модель быстрого прототипирования и эволюционная.

**1.4 Инструменты разработки**

Для разработки данного проекта будет выбрана среда Unity Engine , так как это удобная и проверенная временем среда разработки. Unity это кроссплатформенная среда разработки компьютерных игр[3], разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями

**2 Проектирование задачи**

**2.1 Разработка UML-диаграмм**

**2.1.1 Диаграмма последовательности**

Диаграмма последовательности UML — такая диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления. Основные элементы диаграммы последовательности это: обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

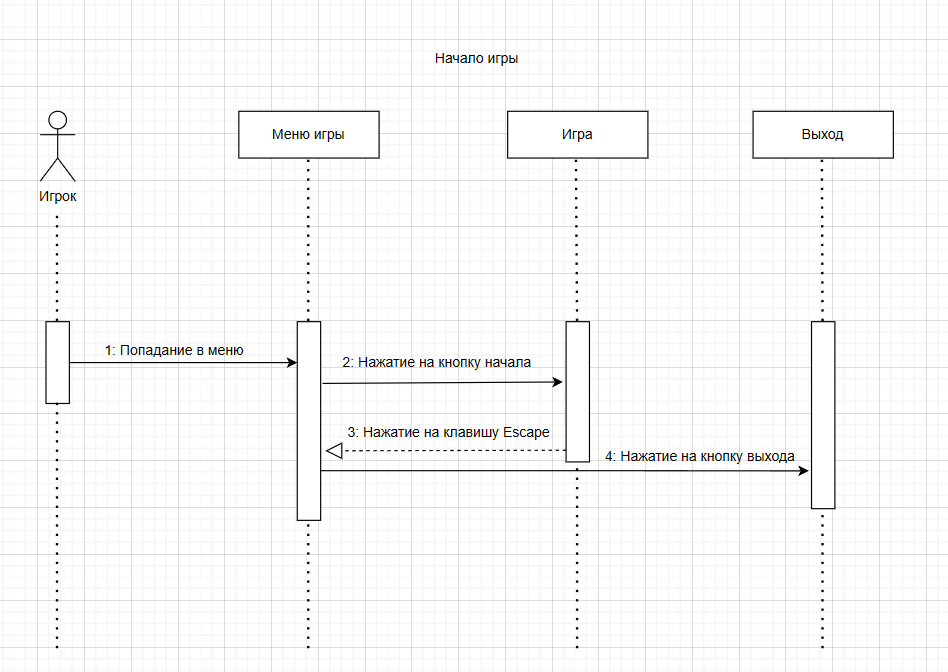
Диаграмма последовательности для проекта показана на рисунке 2.

Рисунок 2 - Диаграмма последовательноости

**2.1.2 Диаграмма классов**

Диаграмма классов – структурная [диаграмма](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_(UML)) языка моделирования [UML](https://ru.wikipedia.org/wiki/UML), демонстрирующая общую структуру иерархии [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) системы, их коопераций, [атрибутов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0) (полей), [методов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), интерфейсов и взаимосвязей (отношений) между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

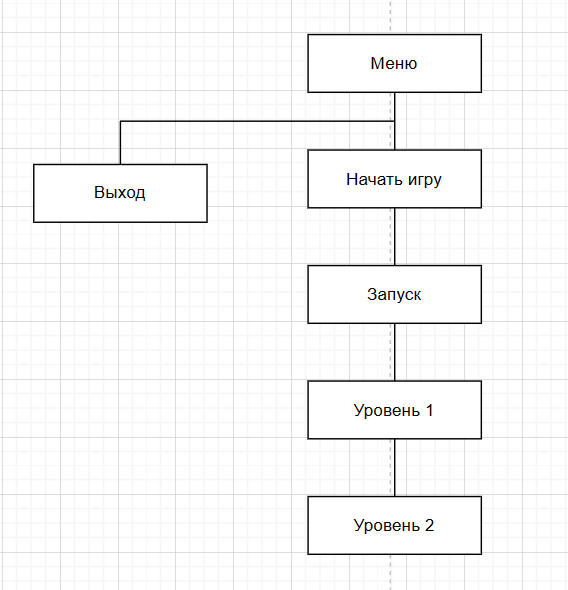
Диаграмма классов для проекта показана на рисунке 3.

Рисунок 3 – Система главного меню

**2.2 Разработка пользовательского интерфейса**

Важным элементом проектирования данного программного продукта является описание внешнего интерфейса разрабатываемого мобильного игрового приложения.

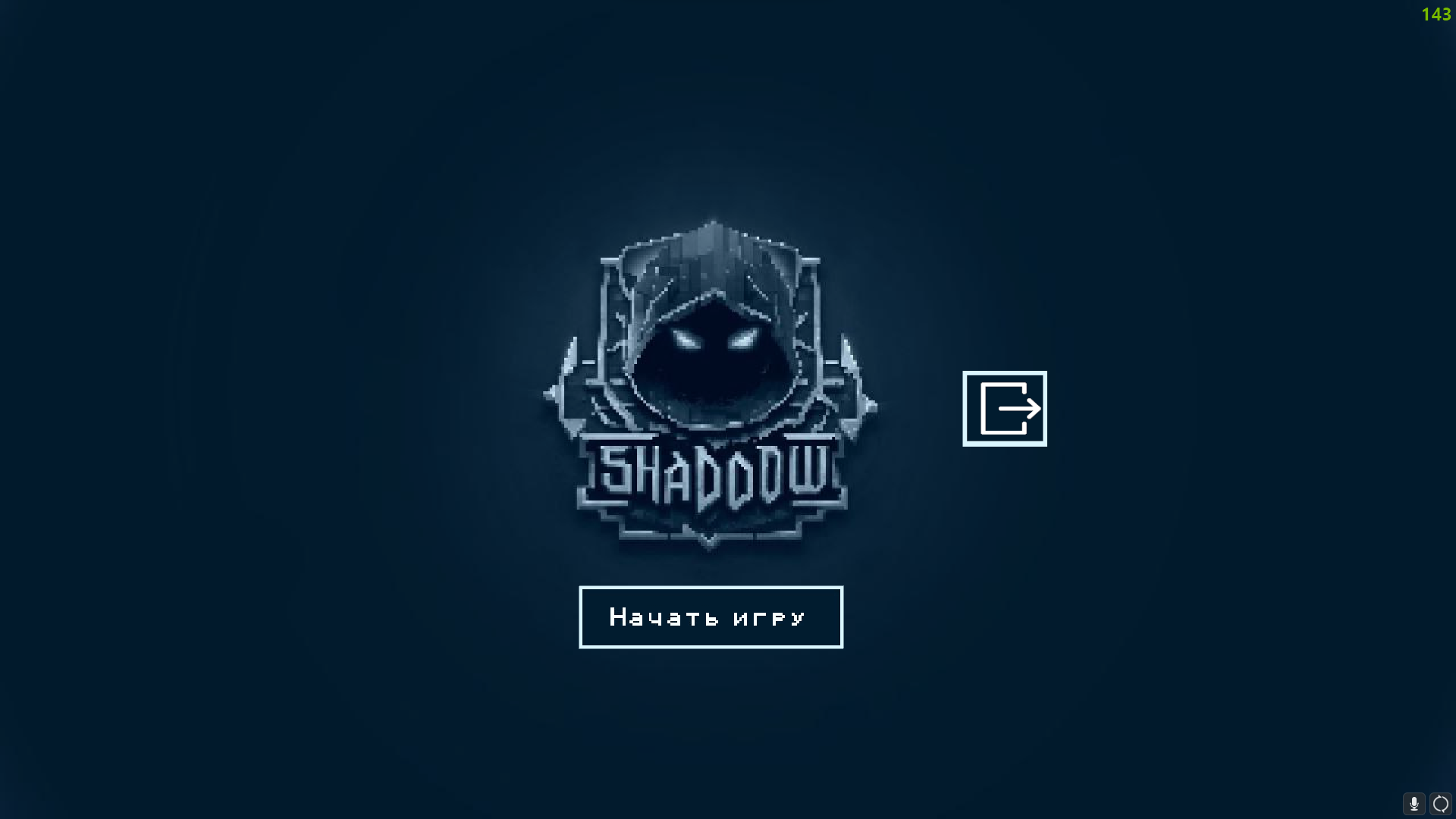
В ходе разработки был спроектирован дизайн главного меню, а также дизайн карт(показанного на рисунках 4,5,6).

Рисунок 4 – Главный экран в игре “ShaDDow”

Рисунок 5 – Игровое поле в игре “ShaDDow”

Весь дизайн игры выполнен в одних и тех же тонах, благодаря этому ничего лишнего на картинке не выделяется.

**2.3 Разработка плана работы над проектом**

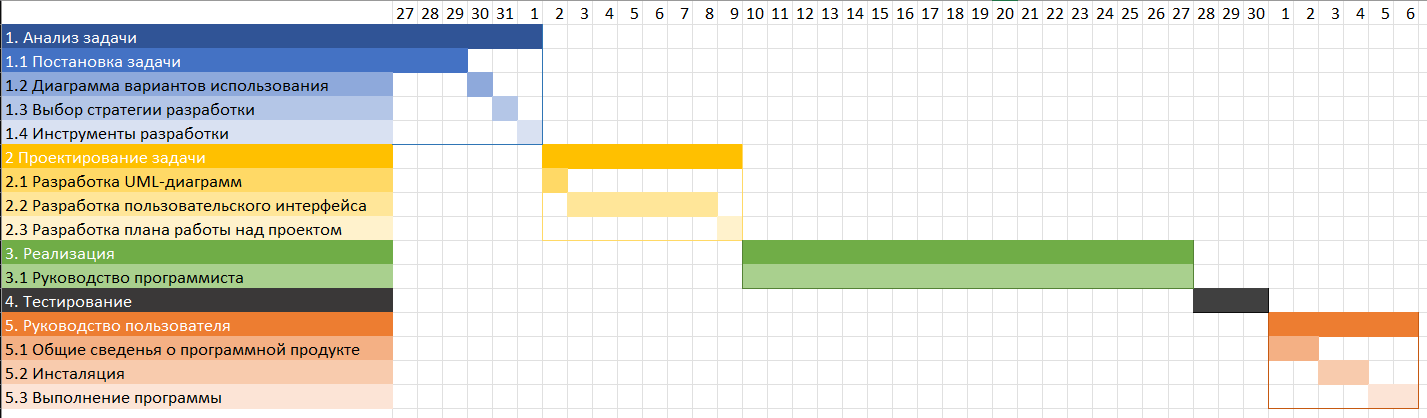
Для разработки плана работы над проектом была создана диаграмма Ганта. Для разработки диаграммы был использован программный продукт Microsoft Excel. Данная диаграмма дает визуальный план работ со сроками выполнения и всеми задачами. Данная диаграмма Ганта представлена на рисунке 7.

Рисунок 6 - Диаграмма Ганта

**3 Реализация**

**3.1 Руководство программиста**

Данный программный продукт был разработан с помощью игрового движка Unity Engine с использованием языка программирования C#. Так как Unity весьма популярный движок и по нему есть множество уроков на всевозможных площадках.

**3.1.1 Реализация Главного меню**

Для создания главного меню необходимо создать сцену. Далее добавить фон, кнопку выхода и входа, добавить на них соответствующие спрайты. Написать скрипты для кнопок, чтобы переключать сцену.

**3.1.2 Реализация первого уровня.**

Для создания уровня мне потребовалось:

1.Набор тайлов для Tile Map

2.Несколько слоев фона для получения эффекта parallax’a

3.Коллайдеры

4. Блоки дизайна

5.Персонаж

6. Монстры

7. Платформы

8.Шипы

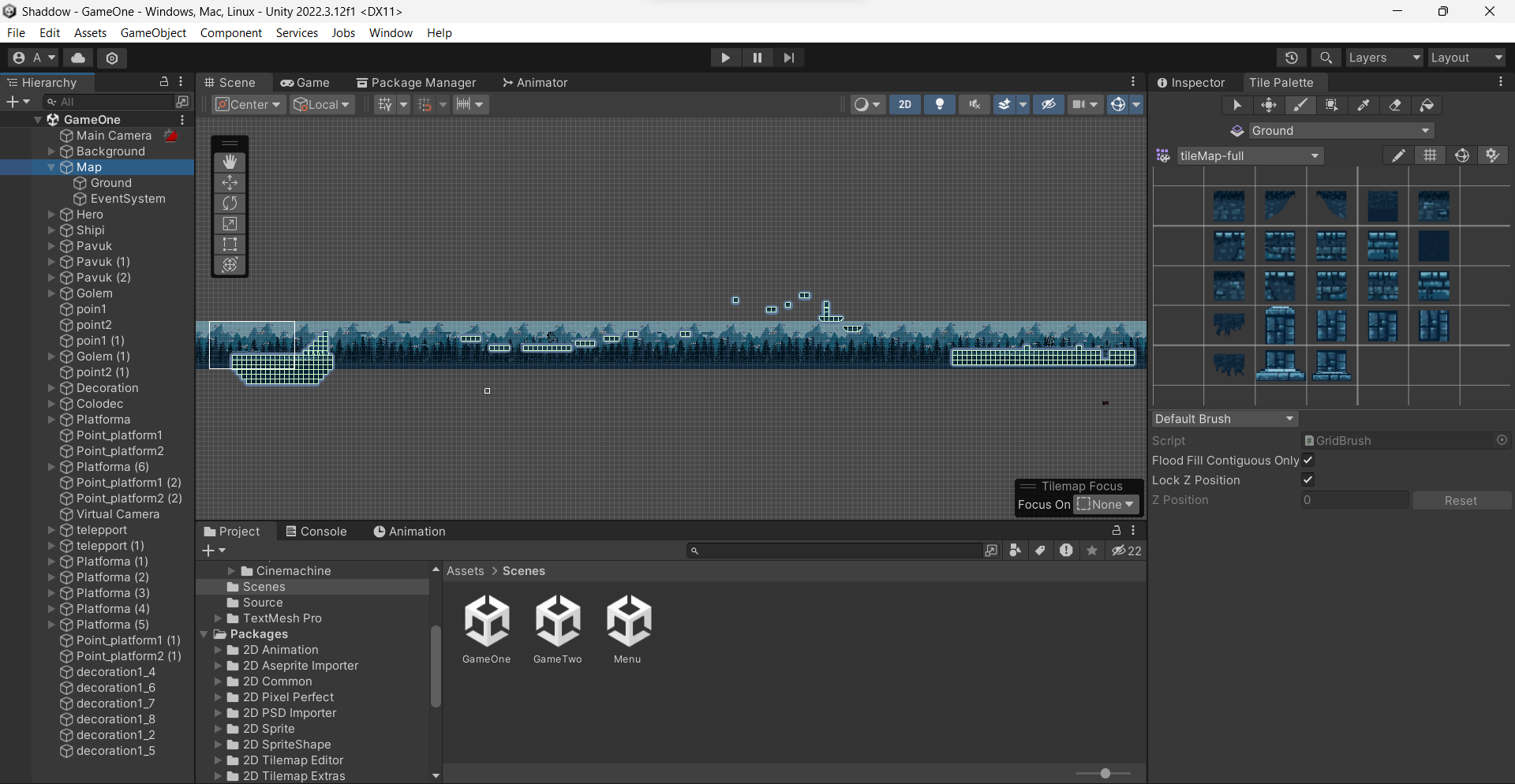
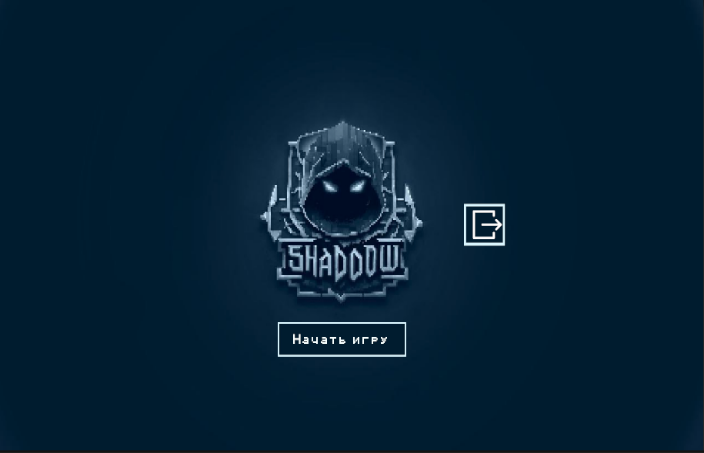
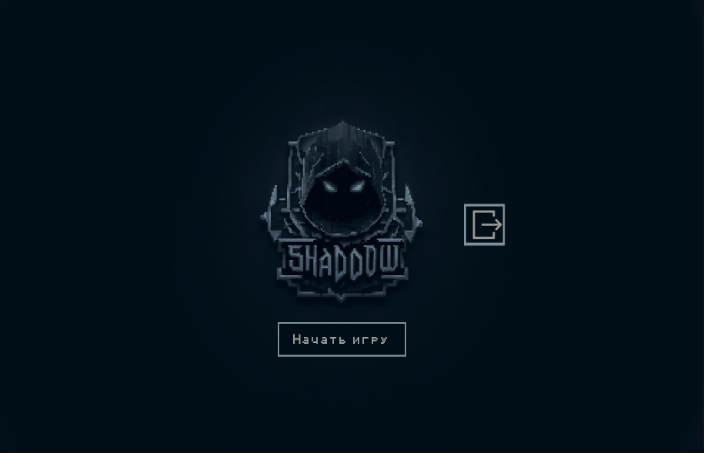
Разработка 1 уровня представлена ниже:

Рисунок 7 – Разработка первого уровня

**3.1.3** **Реализация перехода между главной сценой и первым уровнем.**

Для создания перехода я использовал анимацию затухания сцены, которую реализовал через компонент Animator и событие OnClick. Процесс анимации представлен на рисунках 8, 9, 10.

Рисунок 8 – 1 этап анимации Рисунок 9 – 2 этап анимации

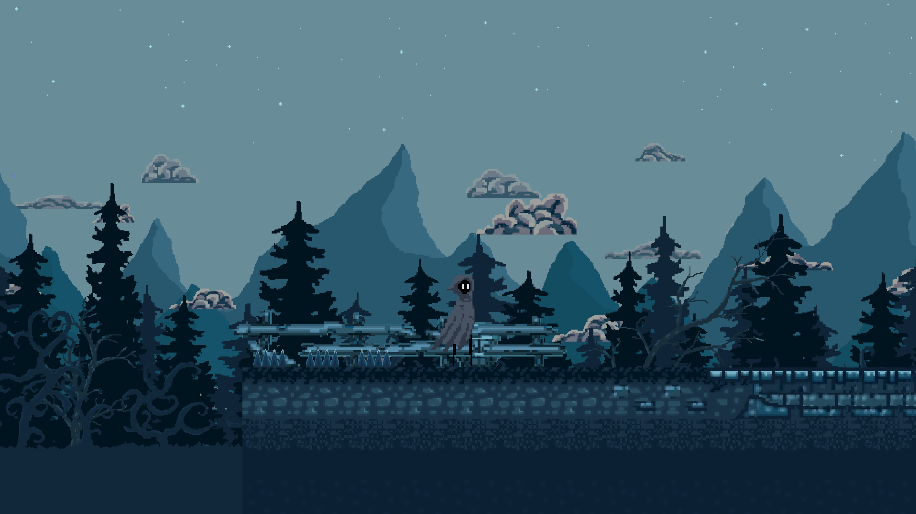


Рисунок 10 – Завершение анимации

**3.1.4** **Реализация второй карты**

Для создания уровня мне потребовалось:

1.Набор тайлов для Tile Map

2.Несколько слоев фона для получения эффекта parallax’a

3.Коллайдеры

4.Блоки дизайна

5.Персонаж

6.Монстры

7.Платформы

8.Шипы

Аналогично первому уровню создается второй. Я его реализовал в виде полигона, где представил все спрайты, механики и анимации.

Пример представлен ниже:

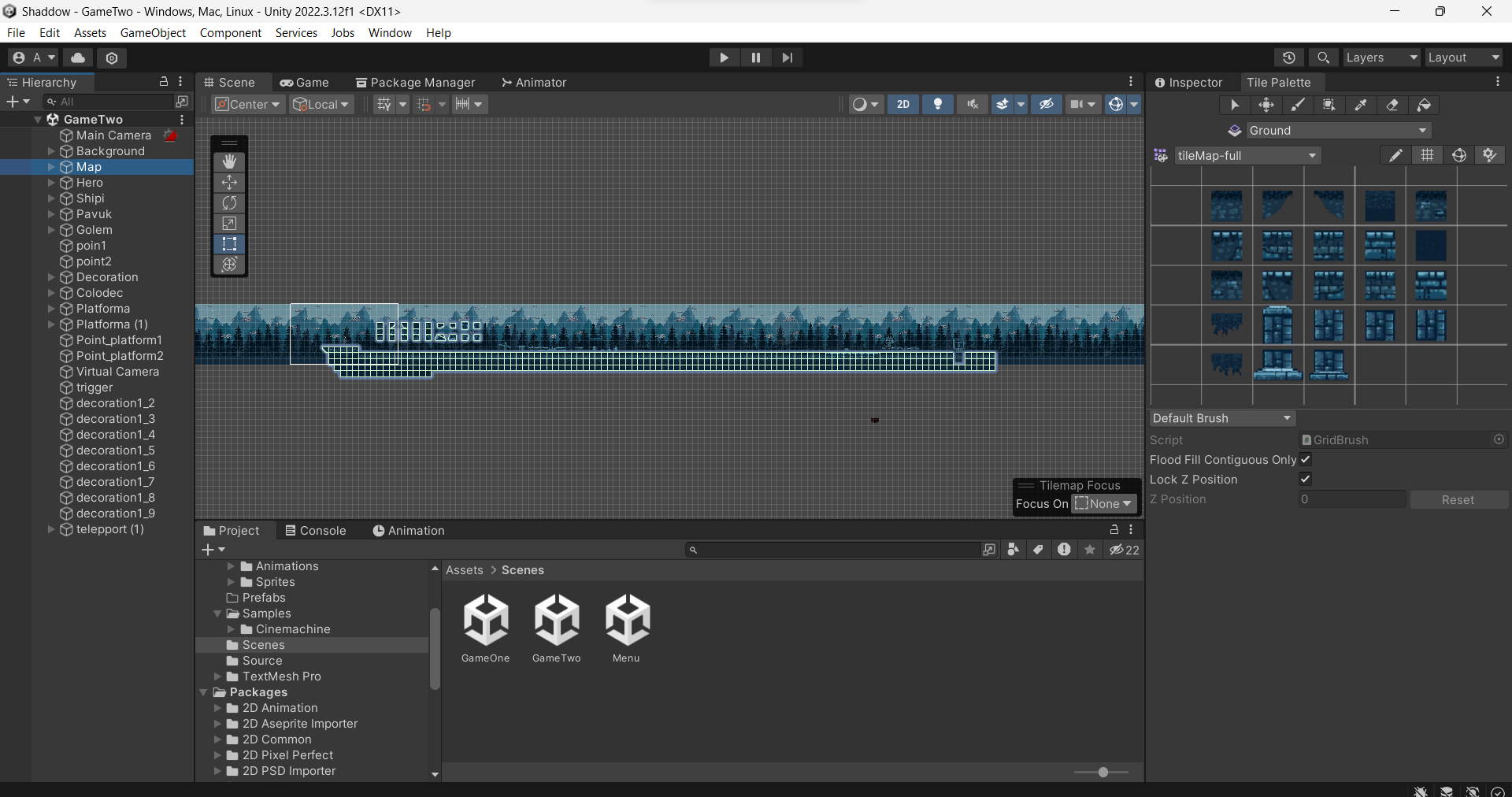


Рисунок 11 – Разработка второго уровня

**3.1.5** **Реализация финала**

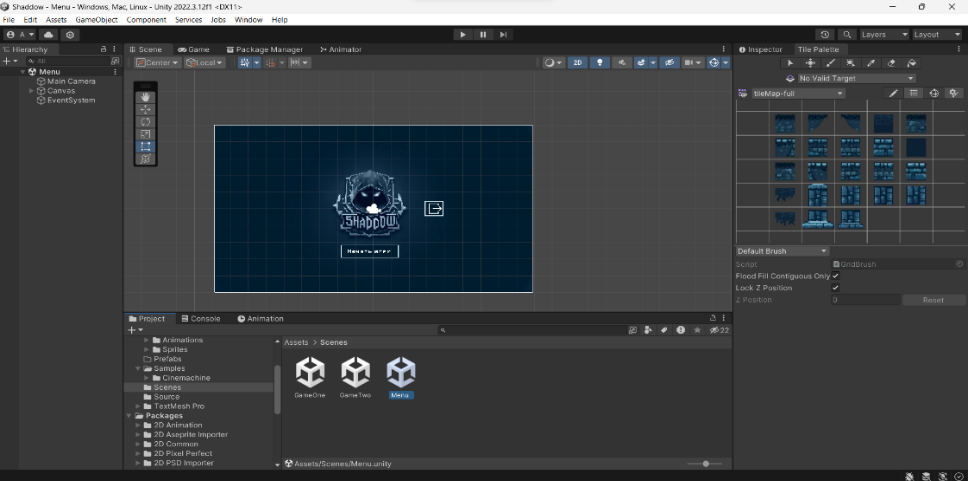
Финал реализован возвратом в главное меню. Для это мне понадобился специальный блок, который реализовывает переход между сценами, если соприкасается с коллайдером главного героя. Такой же принцип используется для перехода между первым и вторым уровнем.

Рисунок 12 – Возврат в меню

**4 Тестирование**

При разработке игрового приложения «ShaDDow» многие возникающие ошибки и недоработки были исправлены на этапе реализации программного продукта. После завершения испытания реализации игрового приложения было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программного продукта в автономном режиме. Тестирование ПП представлено в Таблице 1.

Таблица 1 – Отчёт о результатах тестирования функций для пользователя.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название теста | Действие | Ожидаемый результат | Физический результат | Результат тестирования |
| 1 | 2 | 3 | 4 | 5 |
| Появление главного меню | Зайти в игру | Открытие главного меню | Открылась главное меню | Выполнено |
| Открытие игры | Нажать на кнопку “Начать игру” | Открытие первого уровня | Открытие первого уровня | Выполнено |
| При смерти телепорт в начало уровня | Находится в игре | Телепорт в начало уровня | Телепортировало в начало уровня | Выполнено |
| Получение урона от шипов и монстров | Дотронуться к шипам или монстру | Получение урона | Игрок получил урон | Выполнено |
| Выход на главное окно | Нажать на кнопку “Escape” | Запуск главной сцены | Запустиласть главная сцена | Выполнено |
| Переход на 2 уровень | Прыгнуть в колодец | Телепорт на второй уровень | Телепортировало на второй уровень | Выполнено |
| Нанесение урона монстрам | Нажать клавишу “ЛКМ” | Удар, нанесение урона | Анимация удара, монстр получил урон | Выполнено |
| Уничтожение монстра | Оставить монстра без HP | Анимация смерти, отключение скрипта монстров и их коллайдера | Анимация смерти, отключение скрипта монстра и х коллайдера | Выполнено |
| Переход на финальную сцену(Обратно в меню) | Прыгнуть в колодец на втором уровне | Переход в меню | Перешло в меню | Выполнено |
| Закрытие игры | Нажать на кнопку выхода | Закрытие приложения | Закрытие приложения | Выполнено |

При тестировании программного продукта, в первую очередь нужно обратить внимание на правильную работу всего ПП. Показываются ли все изображения, виден ли текст, нет ли лишних пробелов и больших отступов, идет ли музыка и зациклена ли она и так далее.

В результате проведения тестирования выяснилось, что все ранее оговоренные функции и требования, были разработаны, а также протестированы. Тесты показали, что все функции работают правильно.

**5 Руководство пользователя**

**5.1 Общие сведения о программном продукте**

Название разрабатываемого игрового приложения «ShaDDow».

Данный программный продукт является игрой, предназначенной для веселого проведения времени. Он предназначен для лиц всех возрастов.

Данный ПП занимает не много памяти на диске и является весьма простой программой, так что даже слабый копмьютер сможет установить и запустить его.

**5.2 Инсталляция**

Открытие игры будет производиться через установку архива. После распаковки архива в удобную пользователю папку нужно запустить файл “ShaDDow” с разрешением .exe, после чего приложение готово к работе.

**5.3 Выполнение программы**

Запустить игру можно при помощи способа указанного в пункте 5.2 ПЗ.

После запуска игры вас встречает главная сцена(Меню).

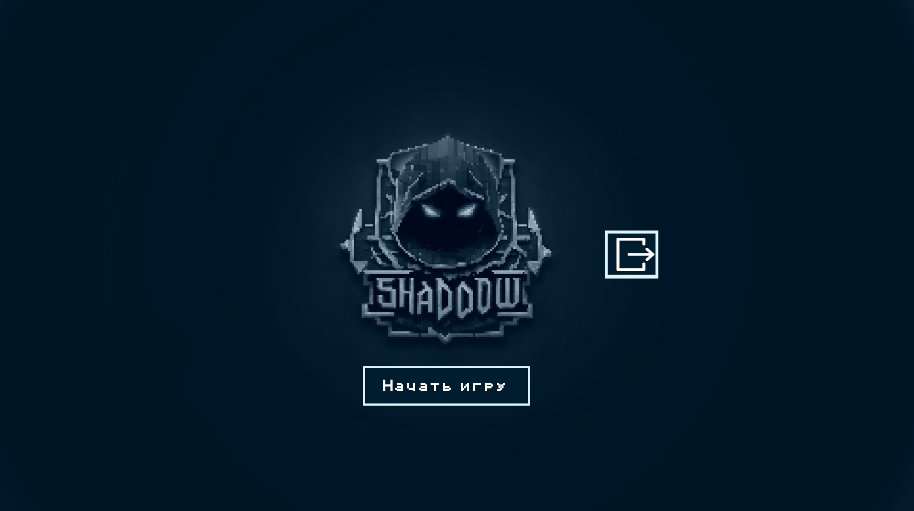
Далее нажмем на кнопку в расположенную прямо под логотипом приложения.

Рисунок 4 – Главный экран в игре “ShaDDow”

После того, как мы запустили игру, мы попадаем на 1 уровень, здесь нам придется пройти полосу препятствий усеянную шипами и монстрами (Уничтожать монстров не обязательное действие, но после их уничтожения уровень становиться намного проще). Если упасть в бездну или получить урон больше 4-ех раз – придется начинать уровень с самого начала .

На 2 уровне, вы можете ознакомиться с тайлами, механиками и анимациями.

Пройдя второй уровень тебя вернет в главное меню, откуда ты можешь попытаться пройти игру еще раз уже с приобретенными знаниями.

**Заключение**

Целью данного проекта на практику являлась разработка игрового приложения «ShaDDow».

Данное приложение оптимизированно под большинство устройств, так как программа очень простая и не имеет графики, которая могла бы напрячь какие-либо процессоры. При желании проект можно перенести даже под Android устройства. Благодаря базовому управлению и простой графике игра предназначена для любых пользователей.

После долгого и тщательного тестирования были выявлены небольшие ошибки и недоработки, которые в последствии были исправлены на стадии тестирования, а также появились идеи для будущих доработок и обновлений. Но не смотря на это приложение уже является работоспособным, и его можно вводить в эксплуатацию.

Так же в процессе создания программного продукта была подготовлена программная документация. Я научился разрабатывать диаграмму Ганта, тесты на использование в процессе тестирования, в процессе разработки познакомился с движком Unity и пополнил свои знания в C#.

**Список использованных источников**

1. Unity, C#**,** Aseprite[https://www.youtube.com](https://www.youtube.com/watch?v=w8rRhAup4kg&t=2855s)
2. Спрайты и их анимации разрабатывались лично в приложении Aseprite.

**Приложение А**

**Листинг программы**

Script Enemy:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Enemy : MonoBehaviour

{

public Animator anim;

public int maxHealth = 100;

int currentHealth;

void Start()

{

currentHealth = maxHealth;

}

public void TakeDamage(int damage)

{

currentHealth -= damage;

anim.SetTrigger("hurt");

if(currentHealth <= 0)

{

Die();

}

}

public void Die()

{

Debug.Log("dead");

anim.SetBool("dead", true);

GetComponent<Collider2D>().enabled = false;

WalkingMonster otherScript = GetComponent<WalkingMonster>();

otherScript.enabled = false;

this.enabled = false;

}

}

Script NewTeleport:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class NewTeleport : MonoBehaviour

{

[SerializeField] private int nextLevelIndex;

private void OnTriggerEnter2D(Collider2D collider)

{

if (collider.gameObject.tag == ("Player"))

{

Debug.Log("Hi");

SceneManager.LoadScene("Menu");

}

}

}

Script exit:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class exit : MonoBehaviour

{

[SerializeField] private int nextLevelIndex;

private void OnTriggerEnter2D(Collider2D collider)

{

if (collider.gameObject.tag == ("Player"))

{

Debug.Log("Hi");

SceneManager.LoadScene("Menu");

}

}

}

Script Obstacle:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Obstacle : MonoBehaviour

{

private void OnCollisionEnter2D(Collision2D collision)

{

if (Hero.Instance != null)

{

if (collision.gameObject == Hero.Instance.gameObject)

{

Hero.Instance.GetDamage();

}

}

}

}

Script WalkingMonster:

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.U2D;

public class WalkingMonster : MonoBehaviour

{

public float speed = 3f;

public Transform[] points;

public int i;

public GameObject childObject;

private Vector3 previousPosition;

public Animator anim;

[SerializeField] public int lives = 8;

private void Start()

{

previousPosition = transform.position;

// Назначение переменной ChildObject для главного объекта

childObject = transform.Find("Square").gameObject;

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject == Hero.Instance.gameObject)

{

Hero.Instance.GetDamage();

lives--;

Debug.Log("У монстра " + lives);

}

if (lives < 1)

{

}

}

private void FixedUpdate()

{

transform.position = Vector2.MoveTowards(transform.position, points[i].position, speed \* Time.fixedDeltaTime);

if (Vector2.Distance(transform.position, points[i].position) < 0.2f)

{

if (i > 0)

{

i = 0;

}

else

{

i = 1;

}

}

Vector3 currentPosition = transform.position;

if (currentPosition.x < previousPosition.x)

{

childObject.transform.rotation = Quaternion.Euler(0, 180, 0);

}

else if (currentPosition.x > previousPosition.x)

{

childObject.transform.rotation = Quaternion.identity; // сбросить поворот

}

previousPosition = currentPosition;

}

}

Script Hero:

using System.Collections;

using System.Collections.Generic;

using System.Runtime.CompilerServices;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Hero : MonoBehaviour

{

[SerializeField] private float speed = 3f; //Скорость передвижения

[SerializeField] public int lives = 5; //Количество жизней

[SerializeField] private float jumpForce = 15f; //Сила прыжка

private Rigidbody2D rb;

private SpriteRenderer sprite;

public Animator anim;

float nextAttackTime = 0;

public float attackRate = 2f;

public Transform AttackPoint;

public float AttackRange = 0.5f;

public LayerMask enemyLayers;

public static Hero Instance { get; set; }

public Animator animator;

private void Awake()

{

if (Instance == null)

{

Instance = this;

}

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

sprite = GetComponentInChildren<SpriteRenderer>();

isRecharged = true;

}

private void FixedUpdate()

{

if (Input.GetButton("Horizontal"))

Run();

if (Input.GetButton("Jump") && Mathf.Abs(rb.velocity.y) < 0.05f)

Jump();

CheckingGround();

walk();

if (Input.GetKeyDown(KeyCode.Escape))

{

SceneManager.LoadScene("Menu");

}

}

private void Update()

{

if(Time.time >= nextAttackTime)

{

if(Input.GetKeyDown(KeyCode.Mouse0))

{

Attack();

nextAttackTime = Time.time + 1f / attackRate;

}

}

}

public Vector2 moveVector;

private void walk()

{

moveVector.x = Input.GetAxis("Horizontal");

anim.SetFloat("moveX", Mathf.Abs(moveVector.x));

rb.velocity = new Vector2(moveVector.x, rb.velocity.y);

}

private void Run()

{

Vector3 dir = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + dir, speed + Time.deltaTime);

sprite.flipX = dir.x < 0.0f;

}

private void Jump()

{

rb.AddForce(transform.up \* jumpForce, ForceMode2D.Impulse);

}

public bool Dead;

public bool onGround;

public Transform GroundCheck;

public float checkRadius = 0.05f;

public LayerMask Ground;

public int attackDamage = 40;

void CheckingGround()

{

onGround = Physics2D.OverlapCircle(GroundCheck.position, checkRadius, Ground);

anim.SetBool("onGround", onGround);

}

public void GetDamage()

{

lives -= 1;

Debug.Log(lives);

if (lives < 1)

{

Dead = true;

SceneManager.LoadScene("GameOne");

}

anim.SetBool("Dead", Dead);

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag.Equals ("Platform"))

{

this.transform.parent = collision.transform;

}

}

private void OnCollisionExit2D(Collision2D collision)

{

if (collision.gameObject.tag.Equals("Platform"))

{

this.transform.parent = null;

}

}

public bool isAttacking = false;

public bool isRecharged = true;

public Transform attakPos;

public float attakRange;

public LayerMask enemy;

void Attack()

{

animator.SetTrigger("Attack");

Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(AttackPoint.position, AttackRange, enemyLayers);

foreach(Collider2D enemy in hitEnemies)

{

enemy.GetComponent<Enemy>().TakeDamage(attackDamage);

}

}

void OnDrawGizmosSelected()

{

if (AttackPoint == null)

return;

Gizmos.DrawWireSphere(AttackPoint.position, AttackRange);

}

}

Script CameraController:

using System.Collections;

using System.Collections.Generic;

using System.Runtime.CompilerServices;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Hero : MonoBehaviour

{

[SerializeField] private float speed = 3f; //Скорость передвижения

[SerializeField] public int lives = 5; //Количество жизней

[SerializeField] private float jumpForce = 15f; //Сила прыжка

private Rigidbody2D rb;

private SpriteRenderer sprite;

public Animator anim;

float nextAttackTime = 0;

public float attackRate = 2f;

public Transform AttackPoint;

public float AttackRange = 0.5f;

public LayerMask enemyLayers;

public static Hero Instance { get; set; }

public Animator animator;

private void Awake()

{

if (Instance == null)

{

Instance = this;

}

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

sprite = GetComponentInChildren<SpriteRenderer>();

isRecharged = true;

}

private void FixedUpdate()

{

if (Input.GetButton("Horizontal"))

Run();

if (Input.GetButton("Jump") && Mathf.Abs(rb.velocity.y) < 0.05f)

Jump();

CheckingGround();

walk();

if (Input.GetKeyDown(KeyCode.Escape))

{

SceneManager.LoadScene("Menu");

}

}

private void Update()

{

if(Time.time >= nextAttackTime)

{

if(Input.GetKeyDown(KeyCode.Mouse0))

{

Attack();

nextAttackTime = Time.time + 1f / attackRate;

}

}

}

public Vector2 moveVector;

private void walk()

{

moveVector.x = Input.GetAxis("Horizontal");

anim.SetFloat("moveX", Mathf.Abs(moveVector.x));

rb.velocity = new Vector2(moveVector.x, rb.velocity.y);

}

private void Run()

{

Vector3 dir = transform.right \* Input.GetAxis("Horizontal");

transform.position = Vector3.MoveTowards(transform.position, transform.position + dir, speed + Time.deltaTime);

sprite.flipX = dir.x < 0.0f;

}

private void Jump()

{

rb.AddForce(transform.up \* jumpForce, ForceMode2D.Impulse);

}

public bool Dead;

public bool onGround;

public Transform GroundCheck;

public float checkRadius = 0.05f;

public LayerMask Ground;

public int attackDamage = 40;

void CheckingGround()

{

onGround = Physics2D.OverlapCircle(GroundCheck.position, checkRadius, Ground);

anim.SetBool("onGround", onGround);

}

public void GetDamage()

{

lives -= 1;

Debug.Log(lives);

if (lives < 1)

{

Dead = true;

SceneManager.LoadScene("GameOne");

}

anim.SetBool("Dead", Dead);

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag.Equals ("Platform"))

{

this.transform.parent = collision.transform;

}

}

private void OnCollisionExit2D(Collision2D collision)

{

if (collision.gameObject.tag.Equals("Platform"))

{

this.transform.parent = null;

}

}

public bool isAttacking = false;

public bool isRecharged = true;

public Transform attakPos;

public float attakRange;

public LayerMask enemy;

void Attack()

{

animator.SetTrigger("Attack");

Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(AttackPoint.position, AttackRange, enemyLayers);

foreach(Collider2D enemy in hitEnemies)

{

enemy.GetComponent<Enemy>().TakeDamage(attackDamage);

}

}

void OnDrawGizmosSelected()

{

if (AttackPoint == null)

return;

Gizmos.DrawWireSphere(AttackPoint.position, AttackRange);

}

}

Script ChangeScenes:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class ChangeScenes : MonoBehaviour

{

public void Scenes(int numberScenes)

{

SceneManager.LoadScene(numberScenes);

}

public void Exit()

{

Application.Quit();

}

}

Script EnemyPavuk:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyPavuk : MonoBehaviour

{

public Animator anim;

public int maxHealth = 100;

int currentHealth;

void Start()

{

currentHealth = maxHealth;

}

public void TakeDamage(int damage)

{

currentHealth -= damage;

anim.SetTrigger("hurt");

if (currentHealth <= 0)

{

Die();

}

}

public void Die()

{

Debug.Log("dead");

anim.SetBool("dead", true);

Pavuk otherScript = GetComponent<Pavuk>();

otherScript.enabled = false;

GetComponent<Collider2D>().enabled = false;

this.enabled = false;

}

}

Script NewLevelTeleport:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class NewLevelTeleport : MonoBehaviour

{

[SerializeField] private int nextLevelIndex;

private void OnTriggerEnter2D(Collider2D collider)

{

if (collider.gameObject.tag == ("Player"))

{

Debug.Log("Hi");

SceneManager.LoadScene("GameOne");

}

}

}

Script Parallax:

using UnityEngine;

public class parallax : MonoBehaviour

{

public GameObject cam;

public float Parallax;

float startPosX;

float startPosY;

void Start()

{

startPosX = transform.position.x;

startPosY = transform.position.y;

}

void Update()

{

float distX = (cam.transform.position.x \* (1 - Parallax));

float distY = (cam.transform.position.y \* (1 - Parallax));

transform.position = new Vector3(startPosX + distX, startPosY + distY, transform.position.z)

}

}

Script Pavuk:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Pavuk : MonoBehaviour

{

[SerializeField] private int lives = 3;

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject == Hero.Instance.gameObject)

{

Hero.Instance.GetDamage();

lives--;

Debug.Log("У паука " + lives);

}

if (lives < 1)

{

}

}

}

Script Platform:

{

float dirX;

public float speed = 3f;

public Transform[] points;

public int i;

private void Update()

{

transform.position = Vector2.MoveTowards(transform.position, points[i].position, speed \* Time.fixedDeltaTime);

if (Vector2.Distance(transform.position, points[i].position) < 0.2f)

{

if (i > 0)

{

i = 0;

}

else

{

i = 1;

}

}

}

}

Script teleport:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class teleport : MonoBehaviour

{

[SerializeField] private int nextLevelIndex;

private void OnTriggerEnter2D(Collider2D collider)

{

if (collider.gameObject.tag == ("Player"))

{

Debug.Log("Hi");

SceneManager.LoadScene("GameTwo");

}

}