Основное задание:

1. Цель работы: Разработать игру и собрать этот проект под платформы Windows.
2. Цель игрока: Дойти до конца уровня.
3. Сцены: Проект должен состоять из четырёх сцен

3.1. Сцена главного меню: Из этой сцены пользователь может перейти к прохождению уровня, воспользоваться меню настроек звука или выйти из игры, закрыв приложение.

3.2. Сцена прохождения уровня: Игра состоит из одного уровня. Персонаж игрока всегда появляется в одном и том же месте. Задача игрока — пройти до конца уровня (выхода). За уничтожение врагов игроку начисляются очки. В случае смерти персонажа происходит переход на «Сцену поражения». В случае достижения конца уровня — переход на «Сцену победы».

3.2.1. Сцена победы: Отображается сообщение о победе, информация о набранных очках, и появляется кнопка «Далее», возвращающая в главное меню.

3.2.2. Сцена поражения: Отображается сообщение о поражении, информация о набранных очках, и появляется кнопка «Далее», возвращающая в главное меню.

1. Интерфейс главного меню: Вы можете оформить и организовать элементы интерфейса сцены главного меню на своё усмотрение, исходя из сеттинга игры.
2. Интерфейс игры: Элементы пользовательского интерфейса в режиме игры вы можете оформить и разместить на своему усмотрению.
3. Персонаж игрока

6.1 Передвигается в плоскости при нажатии WASD.

6.2 Стрельба — нажатие левой клавиши мыши.

6.3 За каждого убитого врага игроку начисляются очки.

6.4 При значении количества боеприпасов 0 стрельба невозможна, несколько секунд происходит перезарядка.

1. Враги с дистанционной атакой: Враг, имеющий стрелковое оружие, начинает стрелять в сторону персонажа игрока когда персонаж находится в зоне действия врага. Вы также можете реализовать для этих врагов режим патрулирования.
2. На уровне можно разместить «аптечки», пополняющие здоровья игрока. После того как персонаж коснулся «аптечки», его здоровье восстанавливается на несколько единиц, а сама «аптечка» исчезает.

Сопроводительная записка:

При создании данного проекта мною было решено создать четкую архитектуру на базе Zenject с учетом условий SOLID, и главное гибкость и расширяемость решений. В массе своей работа не оказалась сложной, в связи с наличием доступной информации в виде книг и открытых ресурсов [unity.com](https://unity.com/), habr.com и прочие. При реализации возникли ряд обстоятельств ранее мною не известных:

1. Не согласованная работа Rigibody объекта совместно с NavMeshAgent, то есть не возможность использовать иные физические свойства объекта под управлением NavMeshAgent, даже с учетом отключения выполнения логики NavMesh. Это определило, что NavMesh как free продукт, является не гибким и не адаптивным в случае управления объектом на который возможны игровые воздействия, как решение, мною видится в последующем попытаться разработать собственный компонент автоматического управления объектами через физические свойства.

2. Малый опыт билдинга игр, выявил ряд ограничений, к примеру невозможность реализации динамического отключения определенного компонента объекта в собранном билде, хотя данная реализация в движке Unity допустима, данный факт, подразумевает использование иных подходов к реализации.

В части интереса при создании проекта, мне доставило огромное удовольствие с момента первого куба в Blender до итоговой реализации в Unity.

Основной подход к реализации данного проекта: использования DI Container как инструмент построения гибкой зависимости от фактического объекта до его логики



Так же отработано решение по генерации объектов посредством фабрик и многократному использованию объектов посредством пулов, что является оптимальным решением в использовании аппаратных ресурсов.

В части обращения к объектам использована реализация в виде массива структур с основным индексатором в виде HashCode или не основным индексатором по типу, что порядком упростило поиск объектов и обращение к ним, как к определённому объекту, так и группе однотиповых объектов. Что исключает системное использование GetComponent и Find, а так же не требует использования ссылочных переменных, слоев, тэгов.

В части единого формата построения класса, было решение придерживаться следующих правил: класс должен проверять готовность к работе через метод SetClass() и состояние булевой isRun, и возможность отключения выполнения методов класса посредством булевой isStopClass. Минимизировать использование UpDate в принципе, контролировать изменения посредством event, единая точка входа-выхода в виде интерфейса-доступа Inject.

Старт проекта:

Блок схемы доступа к компонентам

:



Блок схемы доступа использование префабов в фабрике



Блок схемы организации пулов

