Названия классов	Описание	Паттерны и приницпы ООП
SoundData SoundRepository SoundsManager Отвечают за хранение и управление звуками в игре	**SoundData** представляет собой класс, который хранит данные о звуках. Класс SoundData является **ScriptableObject**, что означает, что он может быть сохранен как отдельный файл в Unity и использоваться в качестве ресурса. В данном случае, использование ScriptableObject для хранения данных о звуках имеет следующие преимущества: • Упрощение управления данными о звуках • Возможность редактировать данные о звуках в редакторе Unity • Автоматическое сохранение данных о звуках • Возможность использования данных о звуках в нескольких сценариях **SoundRepository** представляет собой класс, который отвечает за управление данными о звуках. **SoundsManager** представляет собой класс, который отвечает за воспроизведение звуков в игре. Класс SoundsManager использует класс SoundRepository для получения данных о звуках и воспроизведения их в игре.	**Синглтон (Singleton):** В классе **SoundsManager** используется синглтон-паттерн для обеспечения единственного экземпляра класса в приложении. **Зависимость через конструктор (Dependency Injection):** В классе **SoundRepository** используется зависимость через конструктор для передачи экземпляра SoundData в класс. Это позволяет классу SoundRepository использовать данные из SoundData без прямой зависимости от него.
GameManager Отвечает за управление состоянием игры	Код состоит из следующих основных частей: **Enum GameEvent** - перечисление, которое определяет различные события игры, такие как начало игры, пауза, рестарт и конец игры. Интерфейс **IGameEventHandler** - интерфейс, который определяет метод **HandleGameEvent**, который должен быть реализован классами, которые хотят получать уведомления о событиях игры. Класс **GameManager** - класс, который отвечает за управление состоянием игры. Он имеет методы для добавления и удаления слушателей событий игры, а также методы для управления состоянием игры.	**Синглтон (Singleton):** В классе **GameManager** используется синглтон-паттерн для обеспечения единственного экземпляра класса в приложении. **Паттерн "Наблюдатель" (Observer Pattern):** В классе GameManager используется паттерн "Наблюдатель" для обеспечения возможности уведомления о изменении состояния игры. **Паттерн "Состояние" (State Pattern):** В классе GameManager используется паттерн "Состояние" для обеспечения возможности управления состоянием игры.

Player PlayerController Отвечают за управление игроком в игре	**Player** представляет собой класс, который реализует синглтон-паттерн и отвечает за управление данными игрока, такими как: • Количество патронов • Инвентарь • Оружие • Анимации **PlayerController** представляет собой класс, который отвечает за управление поведением игрока, таким как: • Движение игрока • Стрельба • Подбор предметов • Обновление анимаций Класс PlayerController также реализует интерфейс **ICommandReceiver**, который позволяет ему получать команды от других классов и выполнять соответствующие действия. В целом, код Player и PlayerController обеспечивает гибкую и расширяемую архитектуру для управления игроком в игре.	**Синглтон (Singleton):** В классе Player используется синглтон-паттерн для обеспечения единственного экземпляра класса в приложении. **Паттерн "Команда" (Command Pattern):** В классах **MoveCommand** и **PickupItemCommand** используется паттерн "Команда" для обеспечения возможности выполнения команд без прямой зависимости от класса PlayerController.
Weapon WeaponController Bullet Этот код представляет собой систему управления оружием в игре и позволяет игроку стрелять из оружия, выпуская пули, которые могут наносить урон объектам в игре	Класс **Weapon** определяет собой оружие, стреляет и создает пули. Класс **Bullet** представляет собой пулю, выпущенную из оружия, и управляет ее движением и столкновением с объектами в игре. Класс **WeaponController** управляет оружием, определяя стратегию стрельбы на основе входных данных.	**Делегаты:** Используются делегат **shootAction** для обработки событий стрельбы. В данном случае делегаты используются для вызова методов ShootStarted и ShootCanceled, которые определяют стратегию стрельбы. **Паттерн "Стратегия":** Класс **WeaponController** использует паттерн "Стратегия" для управления оружием, определяя стратегию стрельбы на основе входных данных (нажатие кнопки мыши). Если пользователь нажимает кнопку мыши, то вызывается метод ShootStarted, который устанавливает значение IsShooting в true. Если пользователь отпускает кнопку мыши, то вызывается метод ShootCancceled, который устанавливает значение IsShooting в false. Это позволяет классу WeaponController независимо от конкретного алгоритма стрельбы использовать любой из них. Преимущества использования паттерна "Стратегия" в данном случае:

		• Улучшение гибкости: класс WeaponController может использовать любой алгоритм стрельбы, не меняя свою внутреннюю реализацию. • Улучшение расширяемости: можно легко добавить новые алгоритмы стрельбы, не меняя класс WeaponController. • Улучшение читаемости: код становится более читаемым, поскольку стратегия стрельбы определяется явно и независимо от конкретного алгоритма. **Принцип единственной ответственности SRP:** Каждый класс в коде отвечает за одну конкретную задачу: • Bullet - реализует поведение пули. • WeaponController - управляет оружием. • IWeapon - реализует общий интерфейс для оружия. • Weapon - определяет собой оружие.
Inventory Item BulletItem Этот код представляет собой систему инвентаря в игре	**Inventory:** Этот класс представляет собой инвентарь игрока и позволяет добавлять и удалять предметы из него. На данный момент в инвентарь ничего не добавляется, так как мы получаем только пули в качестве ресурсов, но если мы будем в будущем получать другие предметы, они будут добавлены в инвентарь. **IItem**: Этот интерфейс определяет методы, которые должны быть реализованы классами-предметами. **Item:** Этот класс представляет собой базовый класс для предметов и реализует интерфейс IItem. **BulletItem:** Этот класс представляет собой конкретный тип предмета (пулю) и наследует от класса Item.	**Паттерн "Шаблонный метод":** Класс **Item** реализует метод Collect, который можно рассматривать как шаблонный метод. Этот метод вызывается, когда предмет собирается, и производит необходимые действия. **Паттерн "Композиция":** Класс **Inventory** использует словарь для хранения коллекции предметов **IItem**. **Принцип единственной ответственности SRP:** Каждый класс в коде отвечает за одну конкретную задачу: • IItem - определяет интерфейс для классов Item. • Item - реализует абстрактный класс для предметов. • BulletItem - реализует конкретный тип предмета. • Inventory - управляет инвентарем игрока.
EnemyPool EnemyPoolEditor EnemySpawner Этот код представляет собой систему спавна врагов в игре	Этот код позволяет спавнить врагов в игре с определенной вероятностью. Он использует пул врагов (**EnemyPool**) для хранения групп врагов и их весов, а также класс **EnemySpawner** для спавна врагов в случайных точках на экране.	**Паттерн "Композиция":** Класс **EnemyPool** содержит список групп врагов (enemyGroups), а класс **EnemySpawner** содержит ссылку на пул врагов (enemyPool). Это позволяет создавать сложные объекты из более простых. Также, класс **ZombyEnemy** использует объект **EnemyHealthBarUI.** **Паттерн "Стратегия":** Класс **EnemySpawner** использует стратегию спавна врагов, которая основана на весах групп врагов.

	В данном коде, **ScriptableObject** используется для создания пула врагов, который содержит список групп врагов (enemyGroups). Этот пул врагов используется в игре для спавна врагов. Для каждой группы врагов есть свой вес (вероятность, с которой будет заспавнен именно данный враг).	**Паттерн "Шаблонный метод":** Класс **EnemyPoolEditor** реализует метод **OnInspectorGUI**, который является шаблонным методом для редактирования пула врагов в инспекторе Unity. **Паттерн "Фабрика":** Класс **EnemyPool** можно рассматривать как фабрику, которая создает объекты класса EnemyGroup и добавляет их в список. **Паттерн "Декоратор":** Класс **EnemyPoolEditor** можно рассматривать как декоратор, который добавляет дополнительную функциональность к классу EnemyPool.
SpawnPointVisulizer Этот код представляет собой визуализатор точки спавна (spawn point) в редакторе Unity	Этот код позволяет визуально представить точки спавна в редакторе Unity. Он использует компонент CircleCollider2D для визуализации точки спавна и позволяет настраивать цвет и радиус визуализации.	
EnemyHealthBarUI Enemy ZombyEnemy EnemyController FloatValue Этот код представляет собой систему управления врагами в игре	Класс **EnemyController** отвечает за управление движением и направлением врага. Класс **FloatValue** представляет собой реактивное свойство, которое позволяет отслеживать изменения значения и реагировать на них. Он используется для реализации здоровья врага. Класс **Enemy** является абстрактным. Он имеет методы TakeDamage, Die и DropItem, которые должны быть реализованы в наследниках. Класс **ZombyEnemy** является наследником класса Enemy и реализует поведение зомби-врага. Он имеет свойство Health, которое представляет собой реактивное свойство. Класс **EnemyHealthBarUI** представляет собой пользовательский интерфейс для отображения здоровья врага. Он использует реактивное свойство Health для отслеживания изменений здоровья врага и обновления пользовательского интерфейса.	**Паттерн "Наблюдатель":** Используется в классе **EnemyHealthBarUI** для отслеживания изменений здоровья врага и обновления пользовательского интерфейса. **Паттерн "Реактивные свойства":** Используется в классе **FloatValue** для реализации реактивного свойства, которое позволяет отслеживать изменения значения здоровья и реагировать на них. **Принцип подстановки Лисков:** В данном коде класс **EnemyController** может принимать в качестве аргумента enemy любого наследника **Enemy** **Принцип единственной ответственности SRP:** Каждый класс в коде отвечает за одну конкретную задачу: • FloatValue - хранение и управление значением здоровья. • Enemy - общий интерфейс для врагов. • ZombyEnemy - реализация конкретного типа врага. • EnemyHealthBarUI - отображение индикатора здоровья врага. • EnemyController - управление движением и ориентацией врага.

UIManager PlayerUI ModalWindow ModalWindowController Этот код представляет собой систему управления пользовательским интерфейсом в игре	Класс **UIManager** является основным классом для управления интерфейсом. Он наследует от класса ModalWindowController и реализует интерфейс IGameEventHandler. Этот класс отвечает за отображение различных панелей интерфейса в зависимости от текущего состояния игры. Класс **PlayerUI** представляет собой пользовательский интерфейс для отображения количества патронов у игрока. Он подписывается на событие изменения количества патронов у игрока и обновляет текстовое поле с количеством патронов. Класс **ModalWindow** представляет собой модальное окно, которое можно активировать или деактивировать. Класс **ModalWindowController** является базовым классом для управления модальными окнами. Он содержит массив модальных окон и методы для отображения и скрытия окон.	**Паттерн "Наблюдатель":** Используется в классе **PlayerUI** для отслеживания изменений количества патронов у игрока и обновления текстового поля, а также класс **UIManager** подписывается на изменения в IGameEventHandler. **Паттерн "Композиция":** Используется в классе **UIManager** для управления модальными окнами и отображения различных панелей интерфейса. **Принцип единственной ответственности SRP:** Каждый класс в коде отвечает за одну конкретную задачу: • ModalWindowController - управляет модальными окнами. • UIManager - управляет пользовательским интерфейсом. • PlayerUI - управляет интерфейсом игрока.
ParallaxRoot Отвечает за создание эффекта параллакса в игре	Класс **ParallaxRoot** работает следующим образом: В методе LateUpdate класс ParallaxRoot обновляет позицию фоновых слоев в зависимости от движения игрока. • Если игрок находится в центре экрана (проверяется методом IfPlayerCenteredScreen), класс ParallaxRoot перемещает фоновые слои в зависимости от скорости движения игрока. • Если игрок не находится в центре экрана, класс ParallaxRoot не перемещает фоновые слои для того, чтобы объекты на сцене не перемещались, когда камера не двигается из-за установленных границ. Обычно помимо ParallaxRoot пишутся такие скрипты как ParallaxLayer и ParralaxScale, но в данной игре я упростила и использовала только ParallaxRoot.	