# **Отчёт по игре**

Вербицкий Данила

Ким Богдан

Иневаткин Юрий

35.2

**ThirdPerson Adventure** — это приключенческая игра, где вы управляете харизматичным героем в жанре третьего лица. Мы создали мощную систему управления персонажем с высокой точностью и отзывчивостью, чтобы вы могли полностью насладиться процессом игры.

### **Особенности управления персонажем:**

* **Динамическая камера**: Камера плавно адаптируется под ваш стиль игры, обеспечивая максимальный комфорт и отличный обзор игрового пространства.
* **Продвинутый контроллер движения**: Гибкость управления позволяет легко переходить между бегом, прыжками, перекатами и другими действиями.
* **Точная физика**: Каждое движение героя ощущается естественным и реалистичным благодаря продуманной физике и анимации.

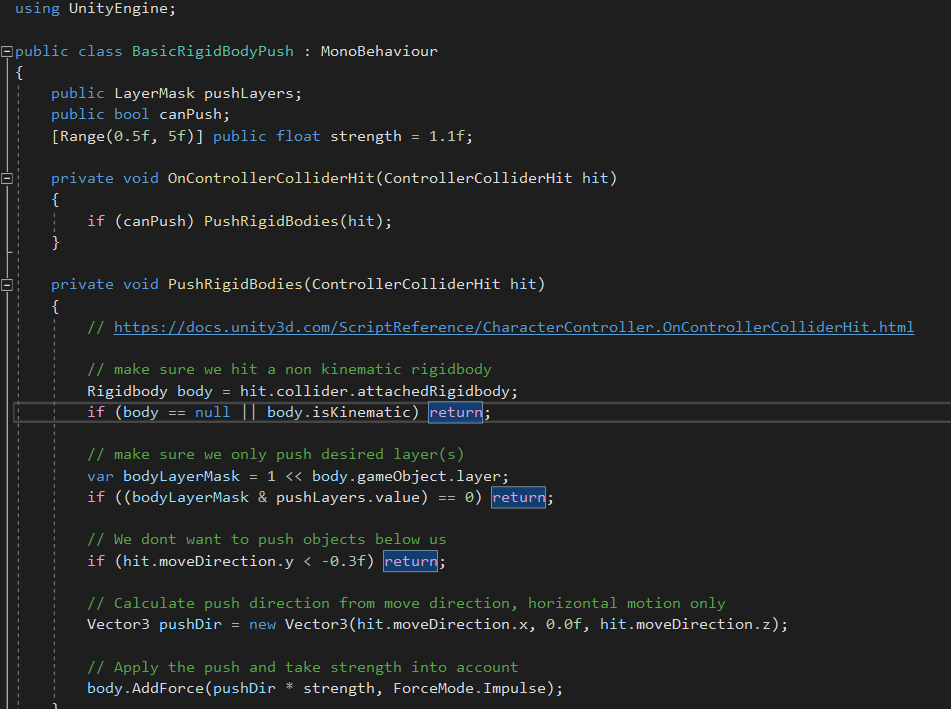
### **Новые обновления:**

* **Гибкость настроек**: Мы добавили новые инструменты для тонкой настройки управления, чтобы вы могли адаптировать игру под свой стиль.
* **Улучшенные анимации**: Движения персонажа стали еще более плавными и детализированными.
* **Интеграция расширенных функций**: Поддержка взаимодействия с окружением — от паркура до использования предметов.

**Технические часть проекта**

Здесь будет описываться самые главные части кода игры и всё, что было необходимо для реализации игры.

1. **Метод реализующий физику взаимодействия с окружением и само взаимодействие с окружением.**



### **Переменные:**

* **pushLayers**: Это маска слоёв, с которыми персонаж может взаимодействовать. В неё можно добавить несколько слоёв, чтобы ограничить, с какими объектами персонаж может взаимодействовать.
* **canPush**: Логическая переменная, которая указывает, может ли персонаж "пихать" объекты. Если эта переменная установлена в false, взаимодействие с объектами не будет происходить.
* **strength**: Множитель силы, с которой персонаж будет "пихать" объекты. Значение варьируется от 0.5 до 5 и определяет, насколько сильно будет применяться сила при столкновении.

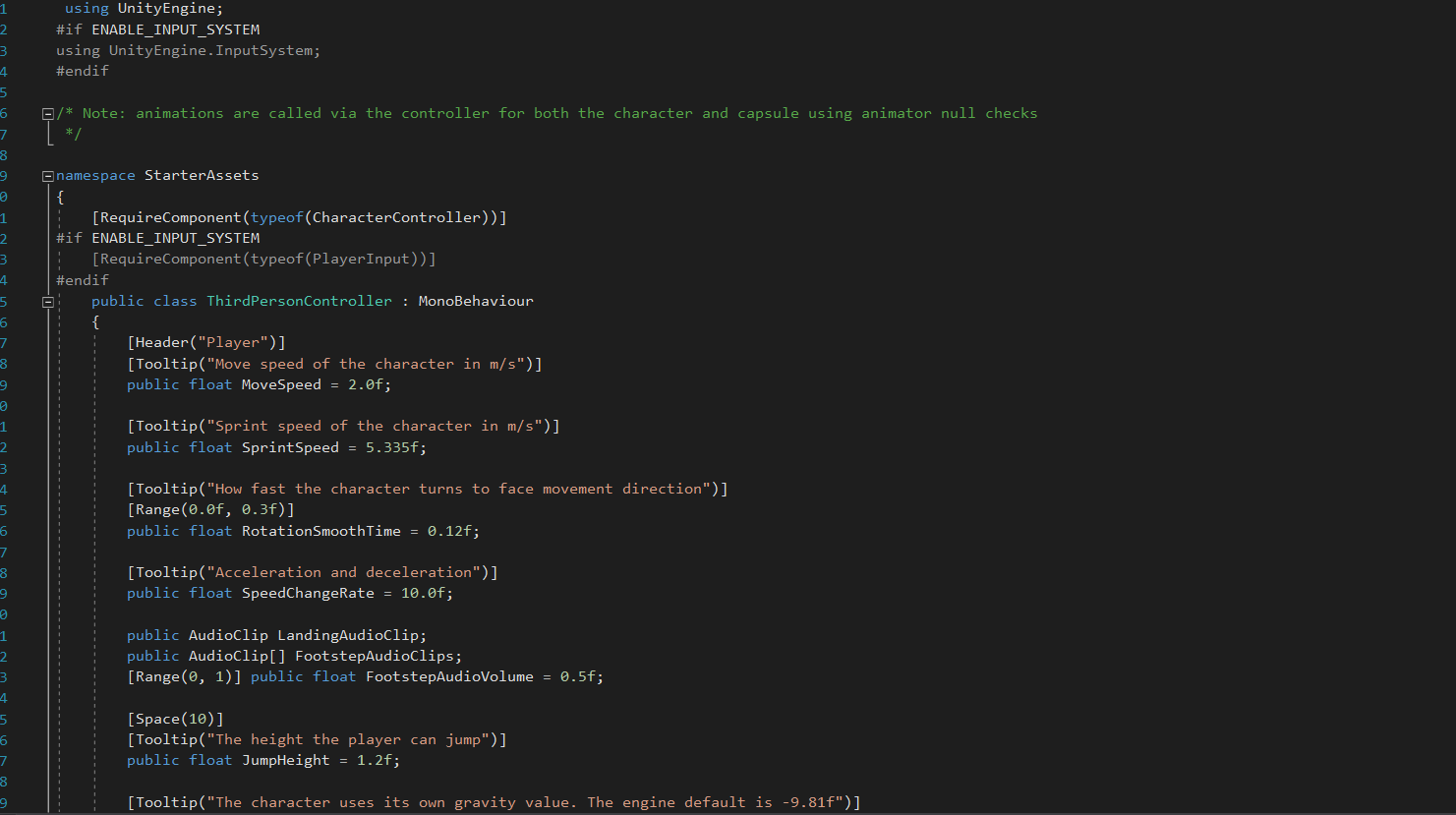
### **Методы:**

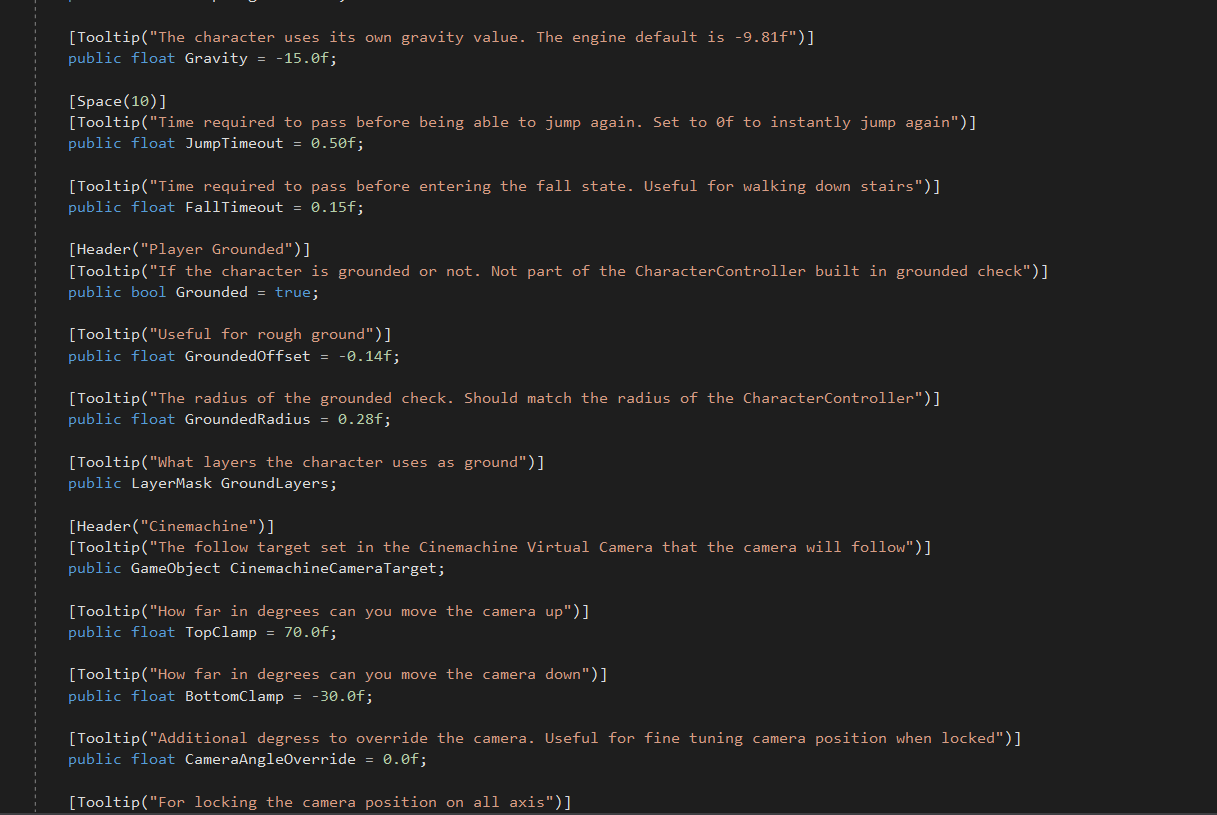
1. **OnControllerColliderHit(ControllerColliderHit hit)**: Этот метод вызывается при столкновении объекта с компонентом CharacterController с другим объектом в игре. В нем проверяется, разрешено ли "пихать" объекты (canPush), и если разрешено, вызывается метод PushRigidBodies(hit), чтобы применить силу к найденному Rigidbody.
2. **PushRigidBodies(ControllerColliderHit hit)**: Это основной метод, который отвечает за взаимодействие с физическими объектами.
   * **Получение Rigidbody**: В первую очередь мы получаем Rigidbody объекта, с которым столкнулся персонаж. Это делается с помощью hit.collider.attachedRigidbody. Если Rigidbody отсутствует или объект имеет флаг isKinematic (что означает, что объект не управляется физикой), то мы игнорируем этот объект.
   * **Проверка слоя**: Далее проверяется, принадлежит ли объект слою, который был указан в маске pushLayers. Если объект не на нужном слое, взаимодействие с ним не происходит.
   * **Игнорирование объектов ниже нас**: Если объект находится ниже персонажа (то есть двигается вниз, а не по горизонтали), то его не "пихаем". Это предотвращает, например, случайные столкновения с объектами, которые падают или двигаются в направлении земли.
   * **Вычисление направления силы**: Направление силы вычисляется только по горизонтальной плоскости (ось X и Z). Мы игнорируем компонент по оси Y, так как нам не нужно "пихать" объекты вверх или вниз, только по горизонтали.
   * **Применение силы**: После всех проверок применяется сила, пропорциональная направлению движения и умноженная на значение strength. Сила применяется с помощью метода AddForce на Rigidbody объекта, используя режим ForceMode.Impulse, что означает мгновенное воздействие силы (как импульс).

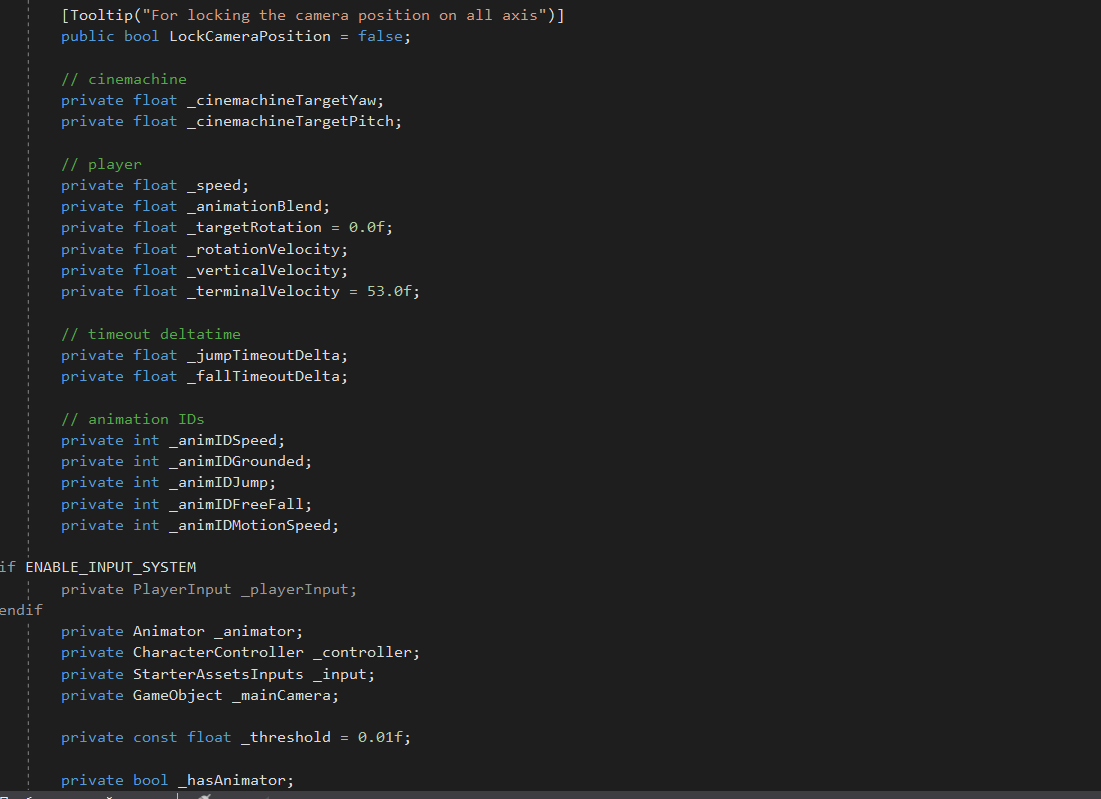
### **Важные моменты:**

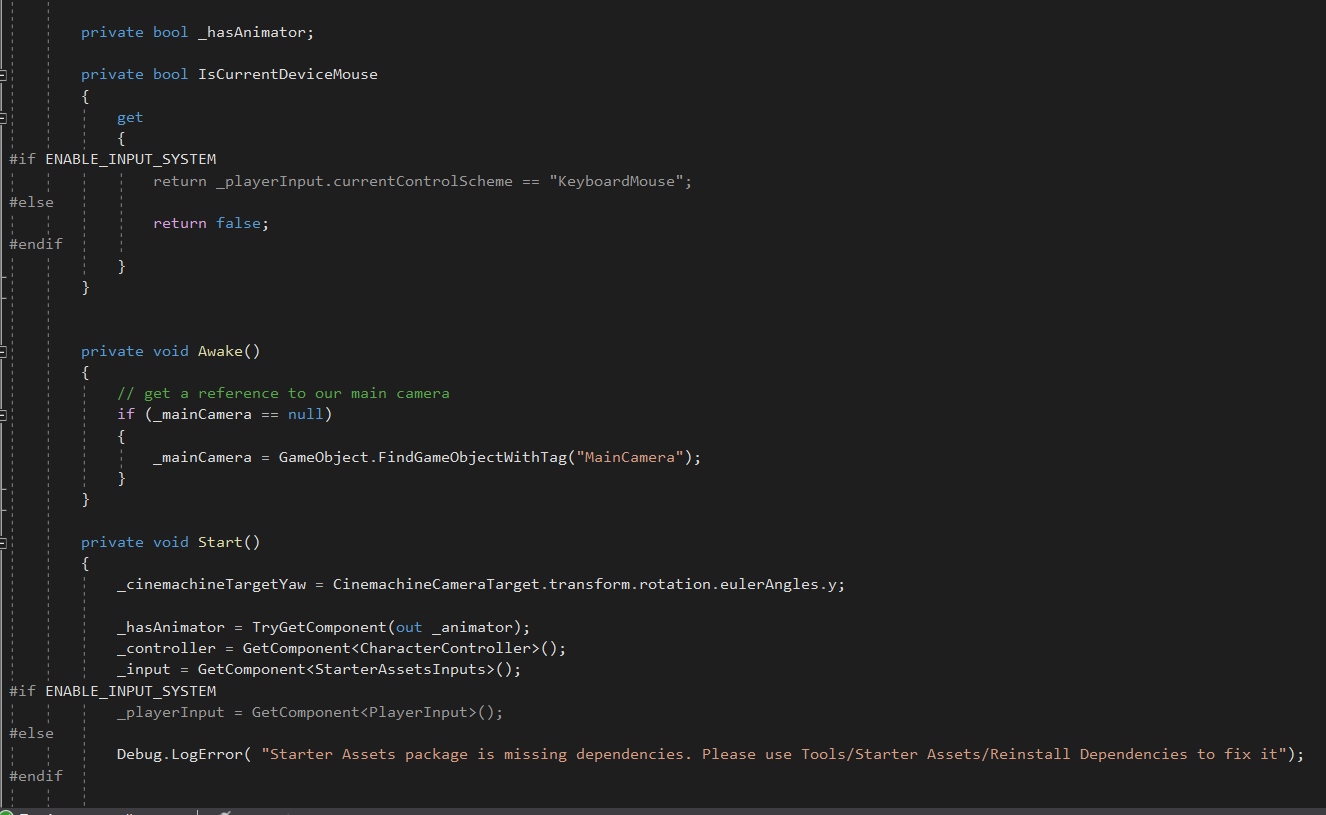
* **Игнорирование объектов с компонентом isKinematic**: Это нужно для того, чтобы не пытаться двигать объекты, которые не управляются физикой (например, те, у которых физика выключена или они неподвижны).
* **Проверка на слои**: Это позволяет точно указать, какие объекты можно пихать, чтобы избежать взаимодействия с теми, которые не должны двигаться в результате столкновения.
* **Игнорирование объектов ниже персонажа**: Это предотвращает применение силы к объектам, которые находятся внизу персонажа, например, если персонаж стоит на платформе и сталкивается с падающими объектами.

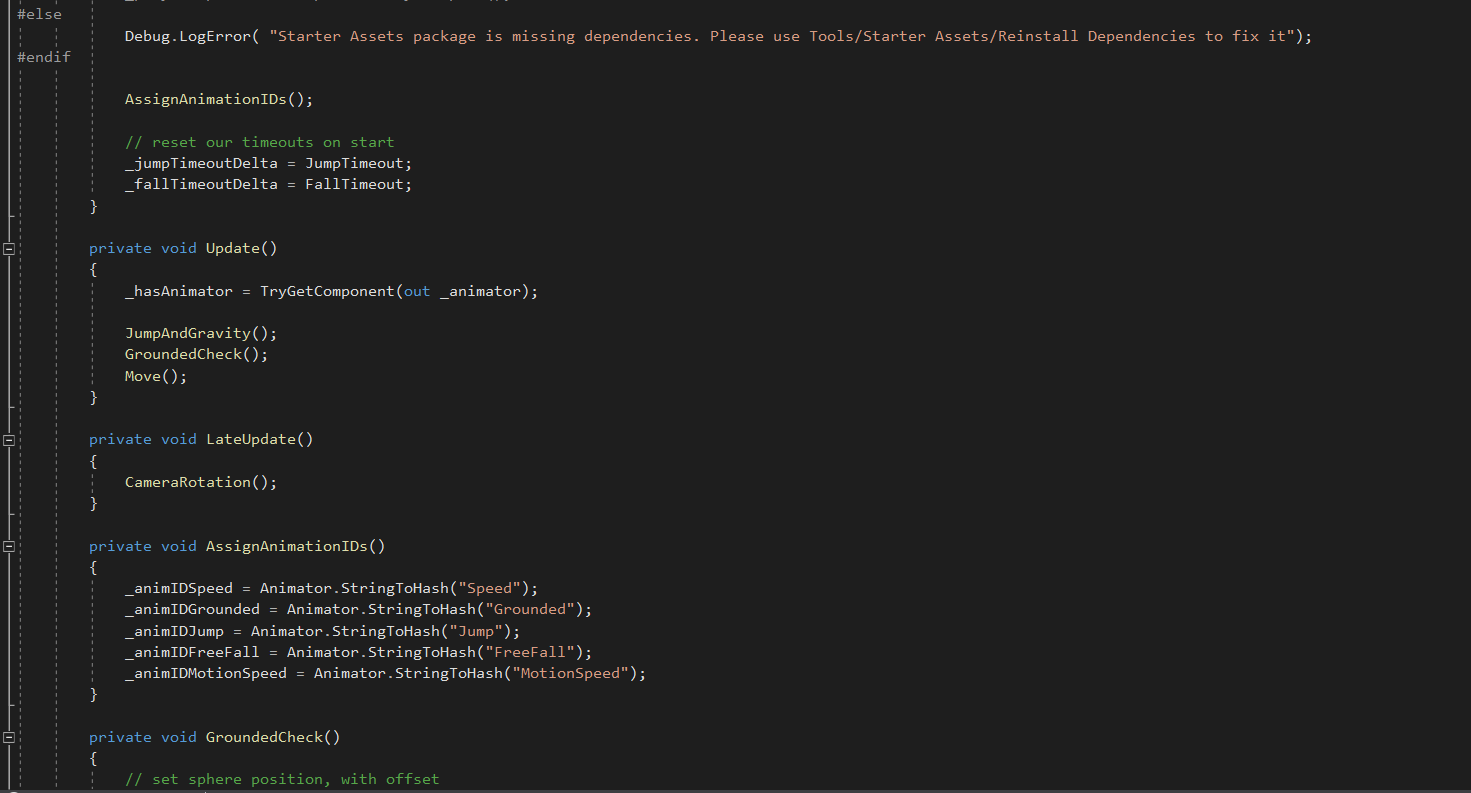
**2. Реализация различного передвижения и взаимодействия с камерой**

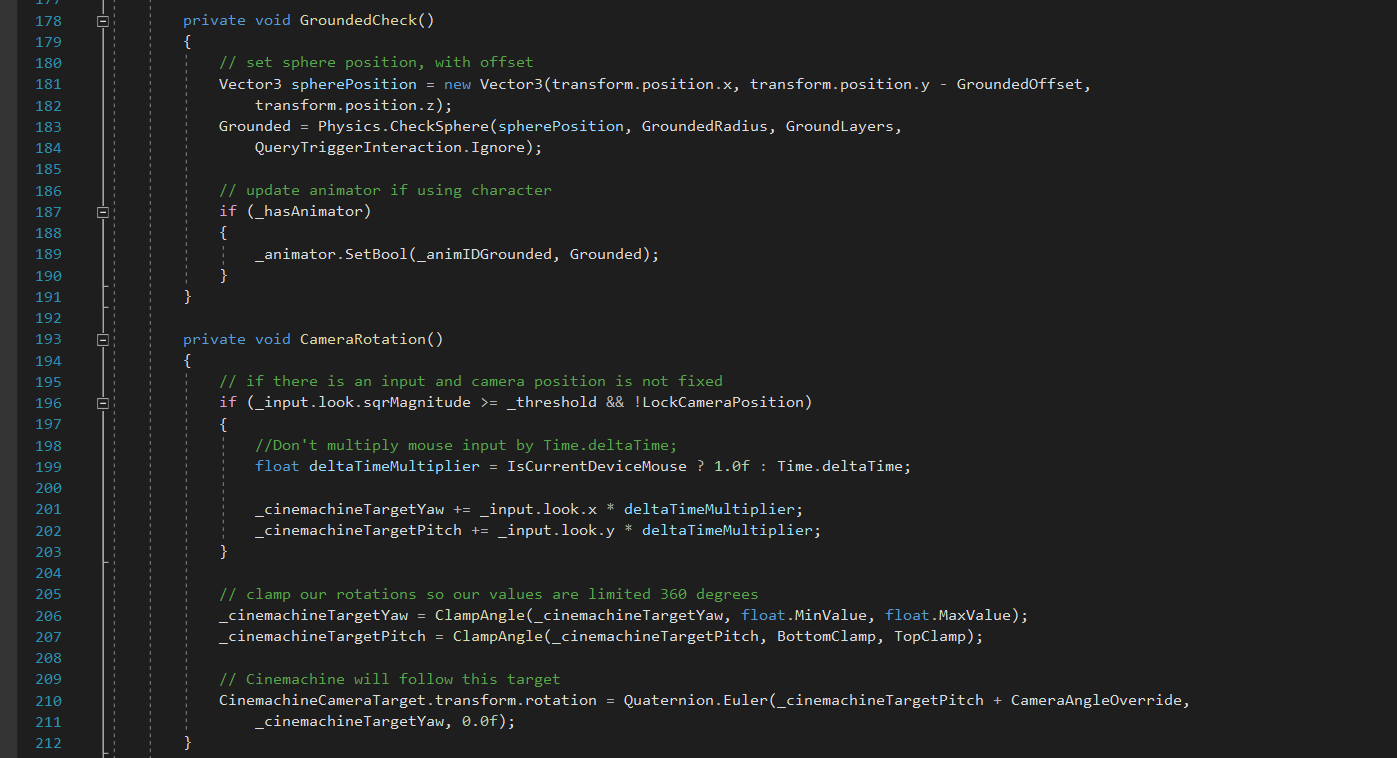
****

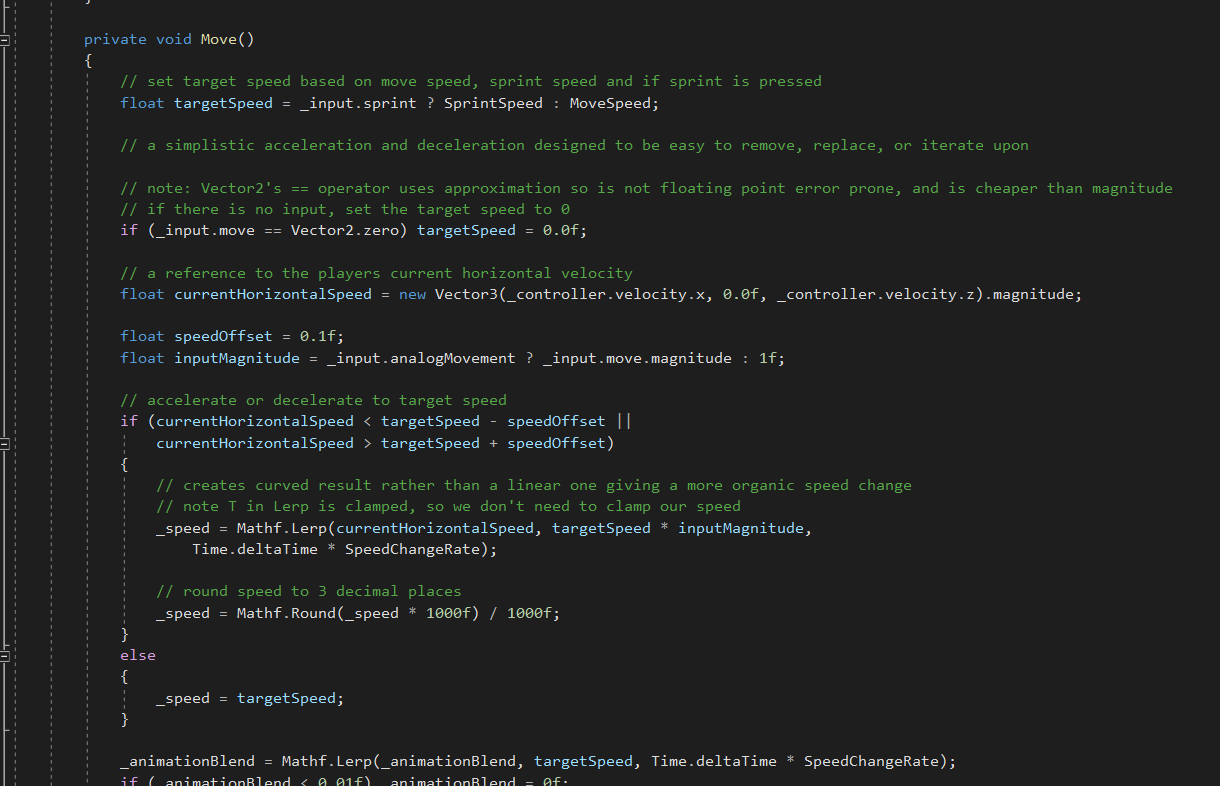
****

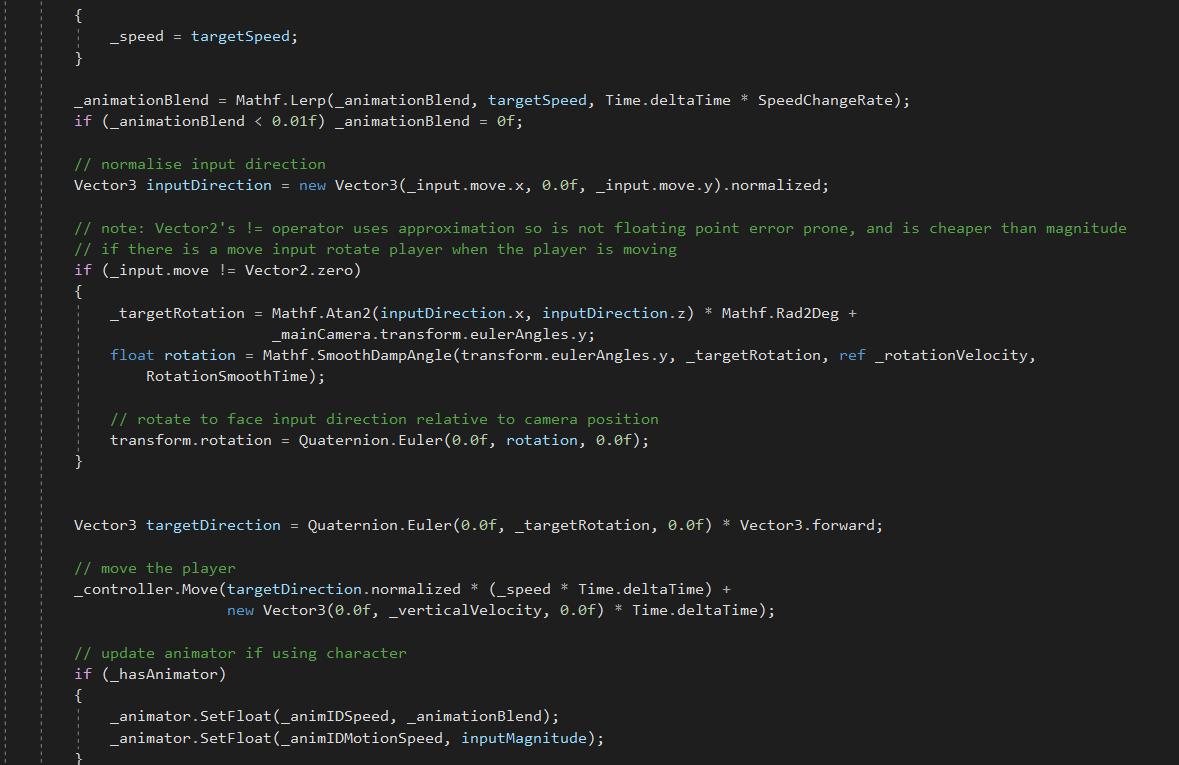
****

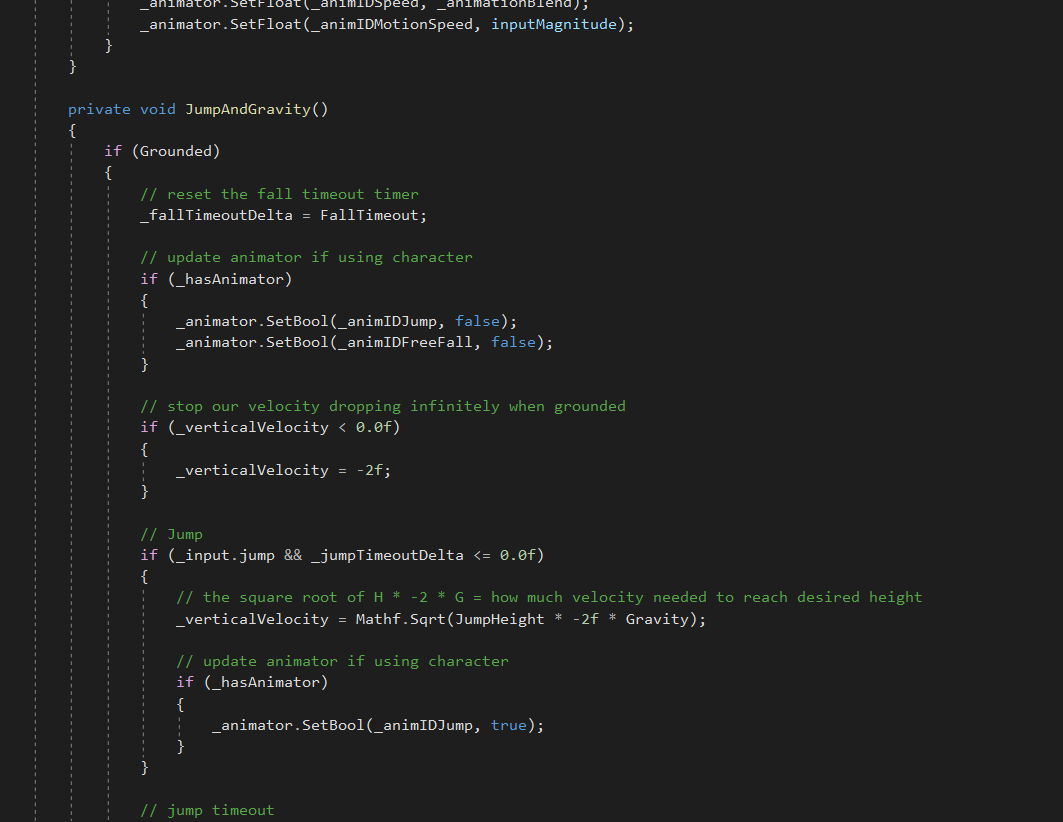
****

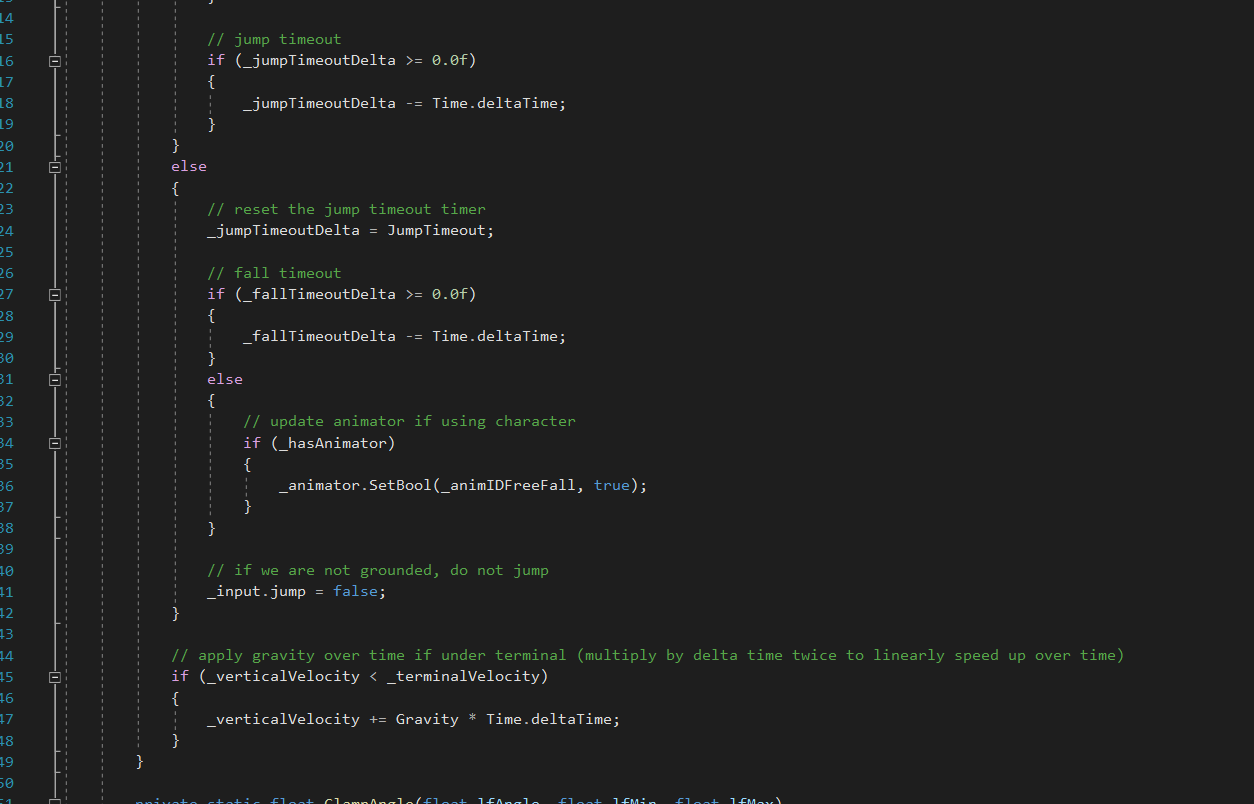
****

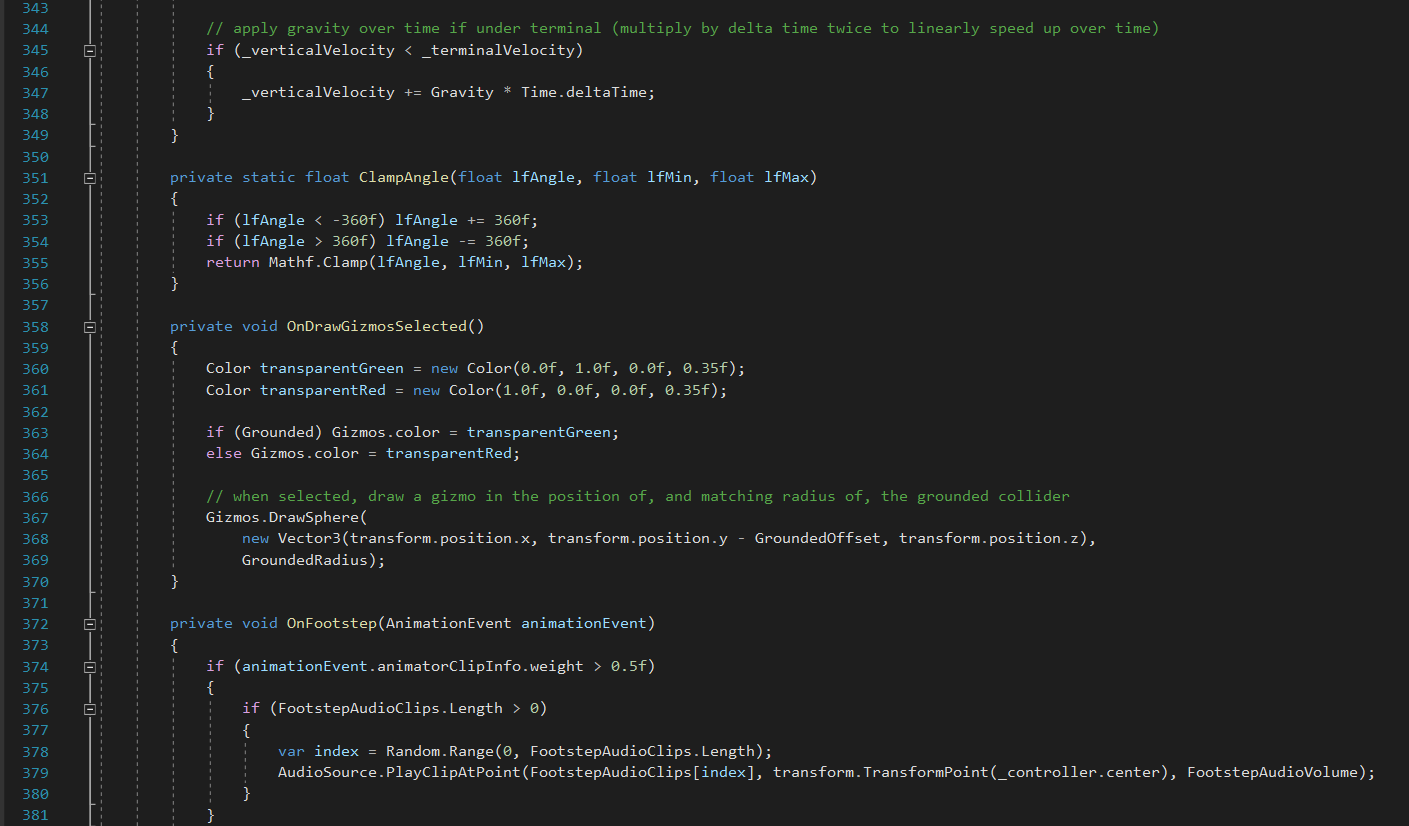
****

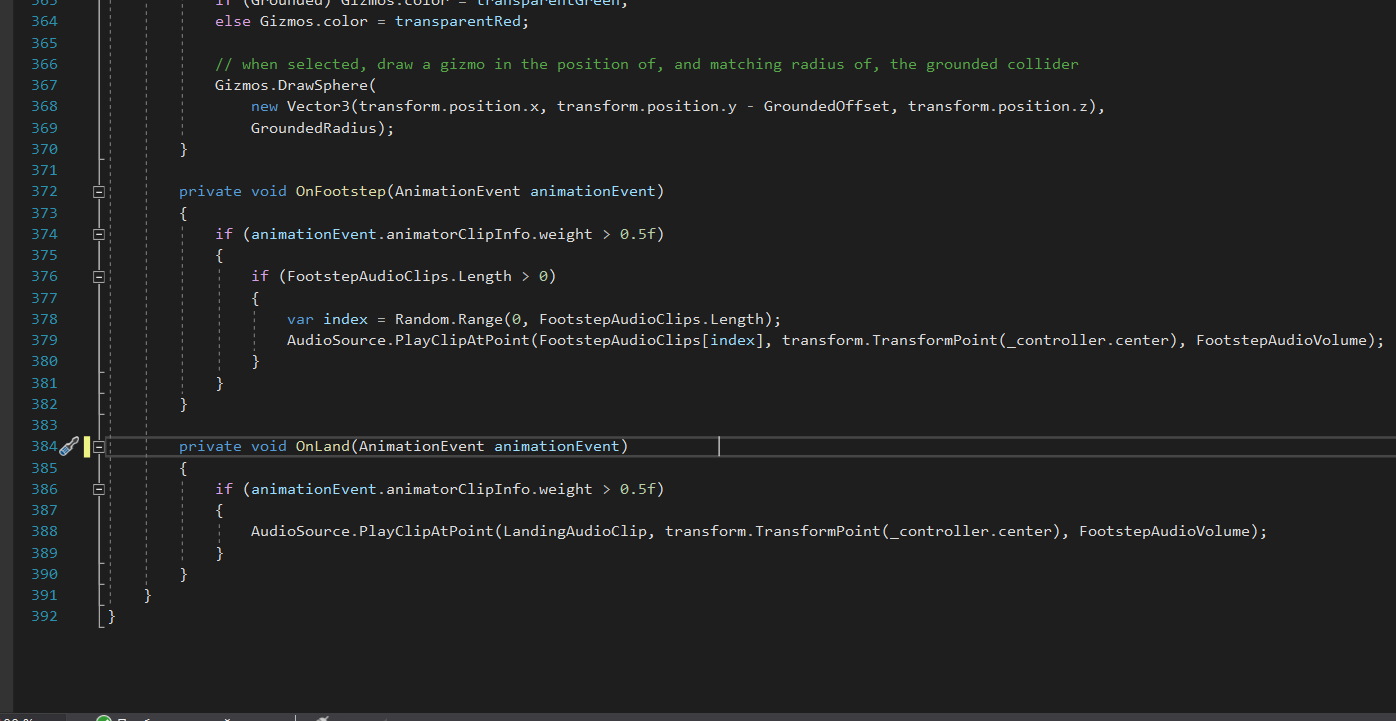
****

****

****

****

****

****

### **Структура и функциональность кода:**

#### **1. Переменные и настройки:**

* MoveSpeed: Скорость передвижения персонажа.
* SprintSpeed: Скорость персонажа при беге.
* RotationSmoothTime: Время для сглаживания поворота персонажа в сторону направления движения.
* SpeedChangeRate: Темп ускорения и замедления.
* JumpHeight: Высота прыжка.
* Gravity: Сила тяжести, применяемая к персонажу (можно настроить для создания "нереалистичных" эффектов).
* Grounded: Флаг, указывающий, находится ли персонаж на земле.
* GroundLayers: Слой, который будет использоваться для проверки на наличие земли.
* CinemachineCameraTarget: Объект, на который будет ориентироваться камера через Cinemachine.

#### **2. Переменные для анимации:**

* Код обрабатывает различные состояния персонажа через анимации, такие как движение, прыжки и падение. Он использует идентификаторы анимации, получаемые через Animator.StringToHash для ускорения работы.
* \_animIDSpeed и другие — идентификаторы для различных состояний анимаций (например, скорость, состояние прыжка, свободное падение).

#### **3. Методы:**

##### **Awake() и Start()**

* Awake(): Здесь выполняется поиск основной камеры и установка значений начальных переменных, включая начало отсчёта таймеров для прыжков и падений.
* Start(): Проверяется наличие компонентов, таких как Animator, CharacterController, и, если используется новый Input System, PlayerInput.

##### **Update() и LateUpdate()**

* Update(): Этот метод используется для обработки логики движения, прыжков и гравитации. В нем вызываются методы для проверки состояния игрока (GroundedCheck()), а также выполняется логика движения (Move()) и прыжков/гравитации (JumpAndGravity()).
* LateUpdate(): В этом методе происходит обновление вращения камеры, чтобы оно происходило после обновления движения персонажа.

##### **GroundedCheck()**

* Проверяет, находится ли персонаж на земле. Для этого используется сфера, которая проверяет, есть ли столкновение с землей, с учетом смещения и радиуса для корректной работы.

##### **CameraRotation()**

* Управляет вращением камеры на основе ввода пользователя. Осуществляется изменение углов поворота камеры с ограничениями на вертикальный угол (от -30 до 70 градусов).

##### **Move()**

* **Основная логика движения персонажа. Здесь:**
  + Рассчитывается целевая скорость в зависимости от того, движется ли игрок или бегает.
  + Управление ускорением и замедлением для плавного перехода между скоростями.
  + Обработка ввода для перемещения и поворота персонажа в направлении движения.

##### **JumpAndGravity()**

* **Управление прыжками и гравитацией:**
  + При нахождении на земле, если пользователь нажимает клавишу прыжка, персонаж получает необходимую скорость для достижения заданной высоты прыжка.
  + Применяется гравитация, если персонаж в воздухе, и он будет падать с ускорением, пока не достигнет терминальной скорости (ограничение на скорость падения).

##### **ClampAngle()**

* Метод для корректного ограничения углов поворота камеры, чтобы избежать значений за пределами ожидаемых (например, углы больше 360 градусов).

##### **OnDrawGizmosSelected()**

* Используется для отрисовки Gizmos в редакторе Unity, чтобы показывать, находится ли персонаж на земле (сфера, которая указывает, где выполняется проверка на наличие земли).

##### **OnFootstep() и OnLand()**

* Эти методы вызываются в момент, когда анимация персонажа достигает определённых точек. Они отвечают за воспроизведение звуков шагов и приземления (используются аудиофайлы, заданные в переменных FootstepAudioClips и LandingAudioClip).

### **Важные моменты:**

1. **Поддержка нового Input System:**
   * Код поддерживает как старую, так и новую систему ввода в Unity. Это проверяется через директиву #if ENABLE\_INPUT\_SYSTEM. Если новый Input System активен, он будет использовать PlayerInput для получения ввода. Если нет, то можно использовать старую систему или его можно отключить.
2. **Анимации:**
   * Анимации контролируются с помощью компонентов Animator. Состояние персонажа (скорость, приземление, прыжок и т. д.) отслеживается и передается в аниматор для переключения состояний.
3. **Механика движения:**
   * Персонаж использует компонент CharacterController для обработки физики движения, что позволяет более точно контролировать столкновения и движения, не используя настоящие физические тела (Rigidbody).
4. **Ротация камеры и персонажа:**
   * Управление вращением камеры и персонажа интегрировано. Камера следит за целевой точкой, обычно за объектом, на котором она ориентирована, и вращается в зависимости от ввода пользователя.

### 

### **Используемые модели и анимации**

В проекте были использованы готовые 3D-модели и анимации, представленные в виде ассетов, которые были адаптированы для нужд нашей игры. Эти ресурсы позволили нам сосредоточиться на основных аспектах разработки, таких как программирование механик и взаимодействие элементов игры, не отвлекаясь на создание базовых визуальных компонентов.

#### **Модели**

Для создания визуальной составляющей игры использовались 3D-модели персонажей и объектов, предоставленные внешними ресурсами. Все модели были тщательно интегрированы в проект с использованием Unity, чтобы обеспечить оптимальную работу в контексте выбранной механики игры.

#### **Анимации**

Анимации персонажей были использованы из готовых наборов, что позволило нам эффективно реализовать динамическое поведение персонажей в игре. Анимации были настроены и адаптированы в рамках Animator Controller, чтобы обеспечить плавный переход между различными состояниями (бег, прыжки, падение и т. д.).

#### **Лицензирование ассетов**

Все использованные ассеты были получены с лицензионных источников, что подтверждает их легальность для использования в учебных проектах. Мы следуем принципам честного использования сторонних ресурсов, полностью соблюдая условия лицензирования и авторские права.

**Примечания**

### 

1. Сосредоточение на программной части: Важным аспектом является то, что использование готовых ассетов позволило нам сосредоточиться на разработке функционала игры (механики, управления), что является основным в рамках нашего задания.
2. Адаптация ассетов: Мы не просто использовали ассеты "как есть", но также провели настройку и адаптацию под специфические нужды нашего проекта (например, настройка анимаций, привязка анимационных событий, изменение материалов и текстур).
3. Учет времени и ресурсов: Такой подход позволил сэкономить значительные временные ресурсы, что важно в рамках ограничений по времени на выполнение задания. В результате, мы смогли создать функциональную игру с соблюдением всех требований к механике.