Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ.**

Специальность: 09.02.07 Информационные системы и программирование

Квалификация: Программист

ОТЧЕТ

ПО ОБЩЕПРОФЕССИОНАЛЬНОЙ ДИСЦИПЛИНЕ: МДК 01.01 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

“Создание проекта, передвижение персонажа.”

Листов: 10

|  |  |
| --- | --- |
| Выполнил студент  Группы П50-4-21  Плахова Анастасия Вадимовна  «\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 года | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_А.Ю.Бушин  «\_\_\_\_» \_\_\_\_\_\_\_2023 года |

Москва 2023

Цель: При помощи базовых знаний интерфейса Unity и языка программирования С#, реализовать передвижение персонажа.

Как только мы открываем приложение нас встречает главный экран, где мы можем посмотреть какие проекты недавно открывали, выбрать какой проект открыть и создать новый проект. Для того что бы создать проект нажимаем на синею кнопку.

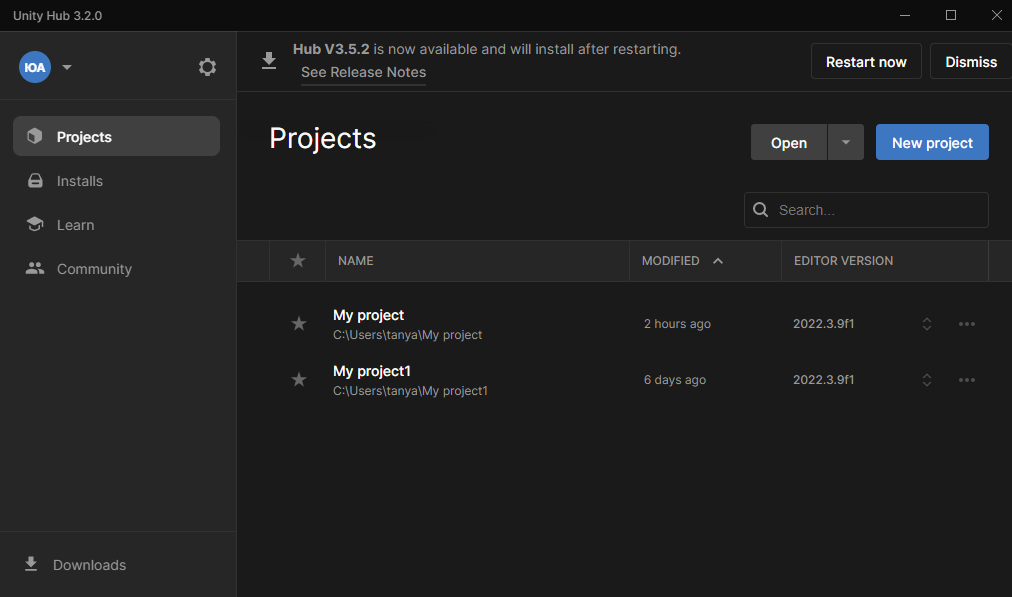


Рисунок главное меню

После перехода мы видим, что можно создать игру в 2D или в 3D, на компьютер или телефон. Нас в данный момент интересует игра, созданная в 3D для компа. Так же можно увидеть, что мы можем выбрать местоположение файла и переименовать его.

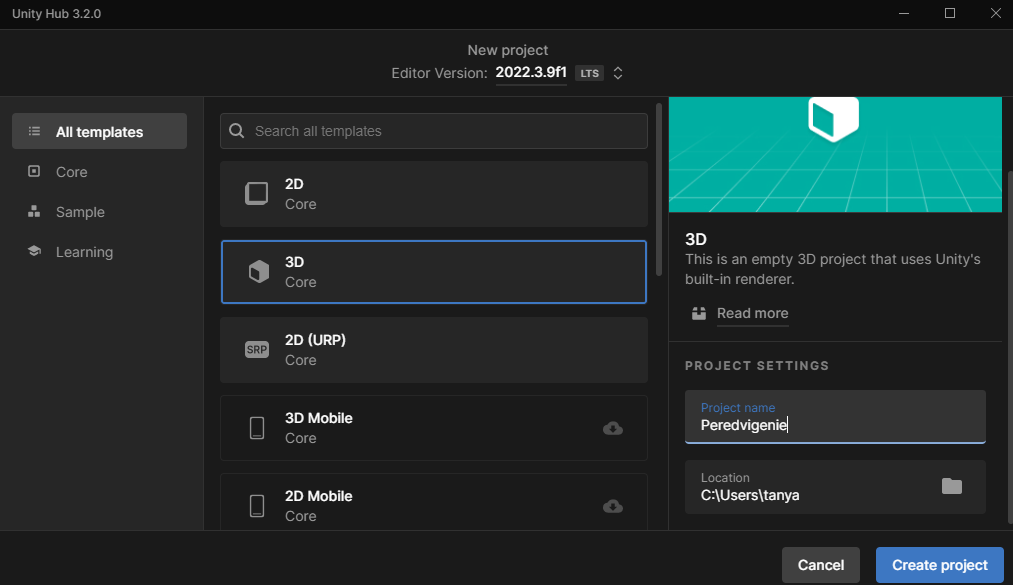


Рисунок создание проекта

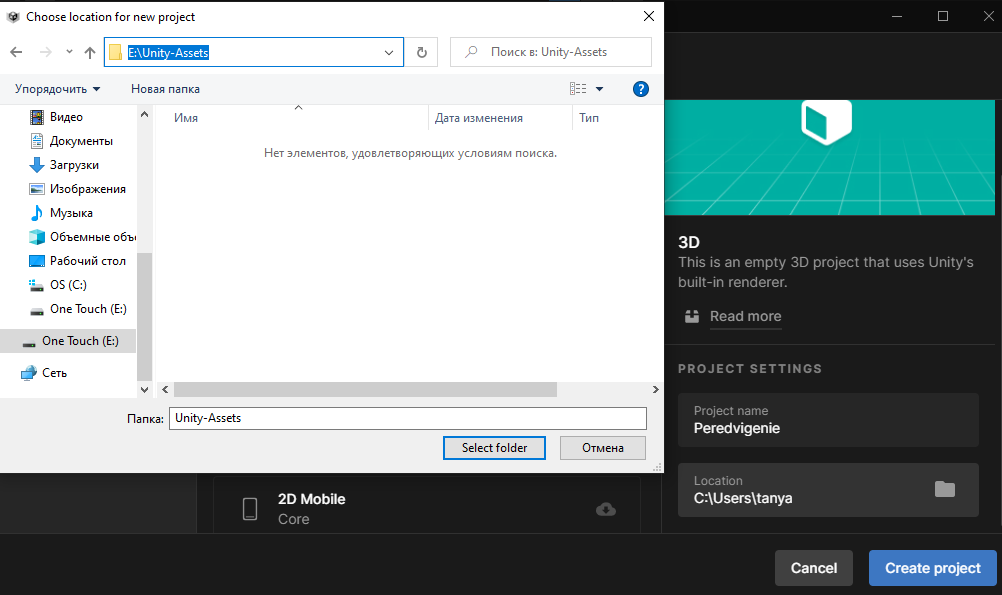


Рисунок выбор места

Теперь же нам надо создать этот проект. После того, как открыли проект мы можем увидеть что у нас на сцене ничего нет.

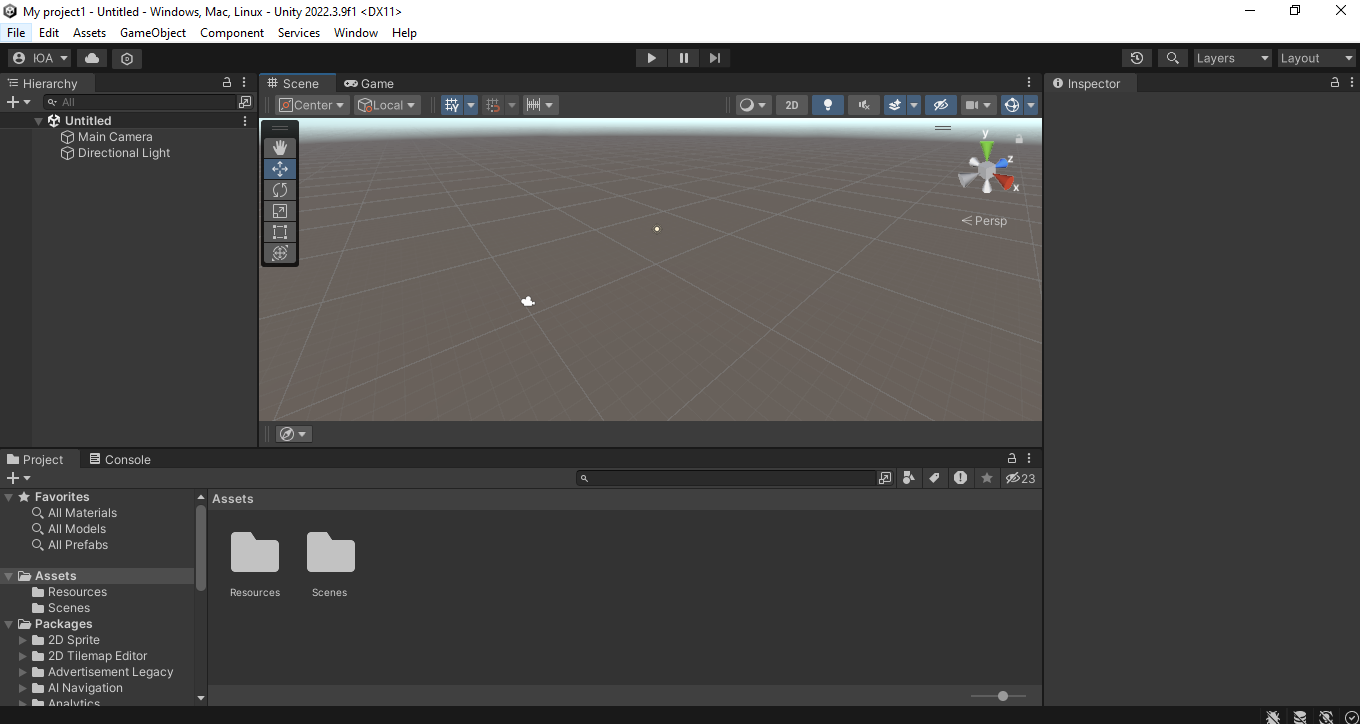


Рисунок первое открытие проекта

Тогда во вкладке Untitled мы должны нажать на правую кнопку мыши и добавить 3D объект и выбрать среди объектов “Plane” и тогда у нас создаться участок, где персонаж будет делать разные действия.

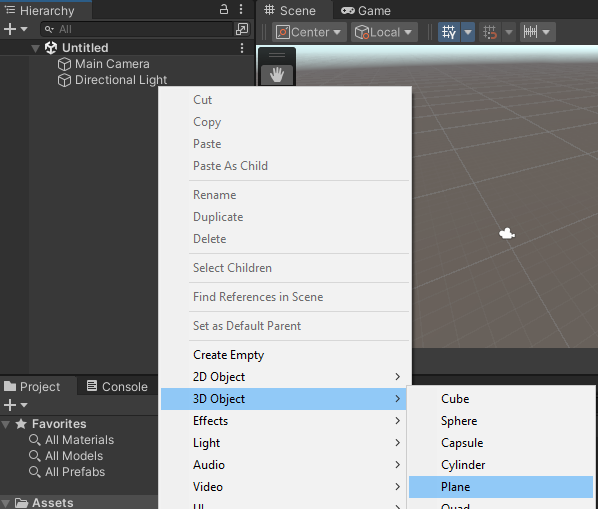


Рисунок создание Plane

После того, как мы создали место для действий, мы должны скачать модель для совершения этих действий. Можно скачать её из библиотеки Unity, где предоставляются бесплатные ассеты. Для этого нам надо нажать “Open in Unity”

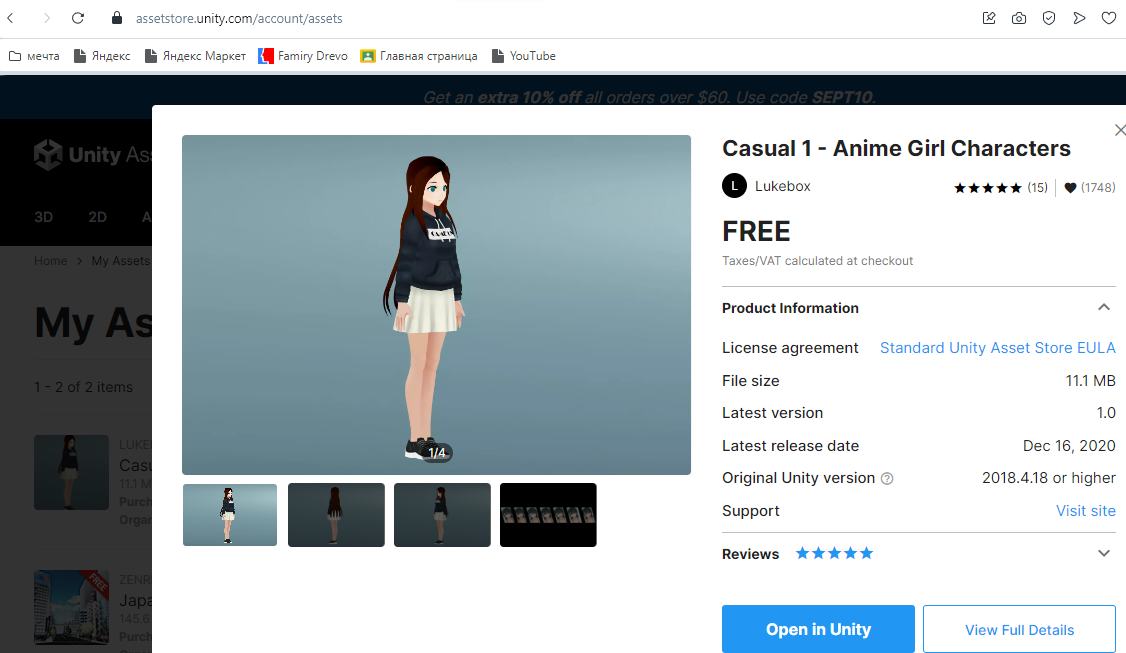


Рисунок модель

Теперь же после этого мы нажимаем на “Import” и наша модель загружается в наш проект. После этого появится ещё одно окно и тут мы нажимаем “Import” и теперь у нас появилась моделькаа.

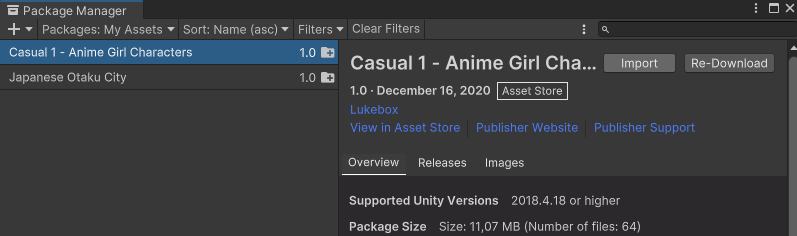


Рисунок загрузка модели

Для перемещения модели на сцену нам надо перетащить её файл с названием “BaseCharactal” на нашу сцену и тогда она у нас появиться и мы можем работать с ней дальше.

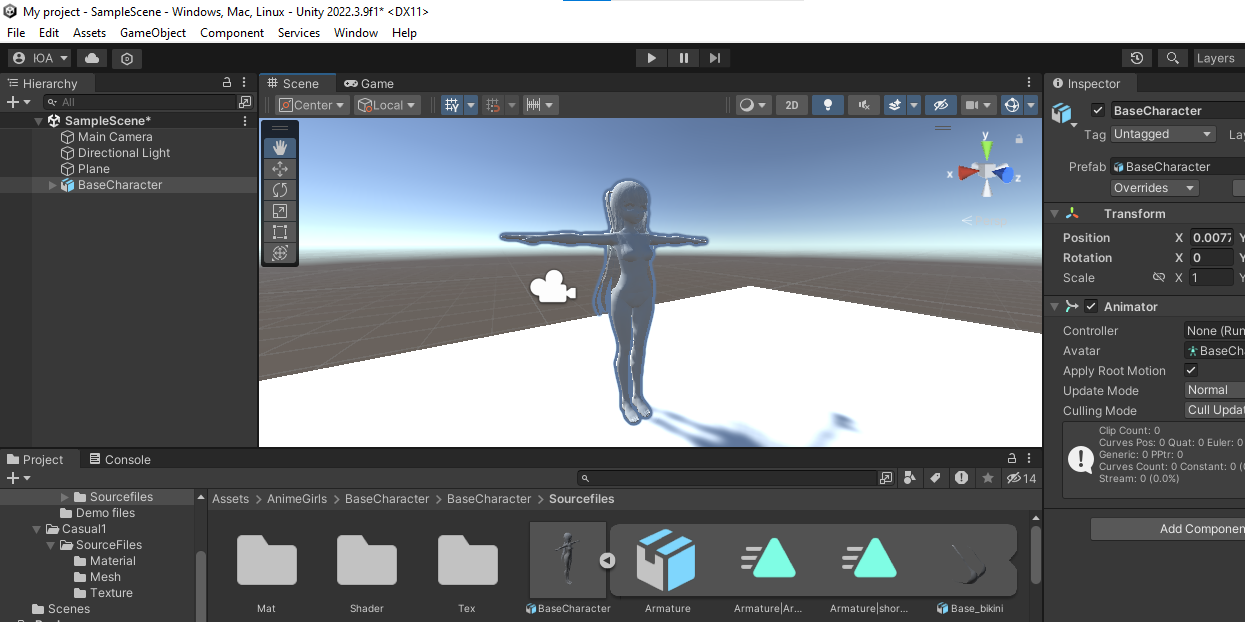


Рисунок создание модели

Далее мы создаём компонент Capsule Collider. Этот компонент определяет область, в которой объект может сталкиваться с другими объектами или реагировать на физические воздействия, такие как столкновения и триггеры. Его размеры можно менять если изменить значения в строчках Radius и Height.

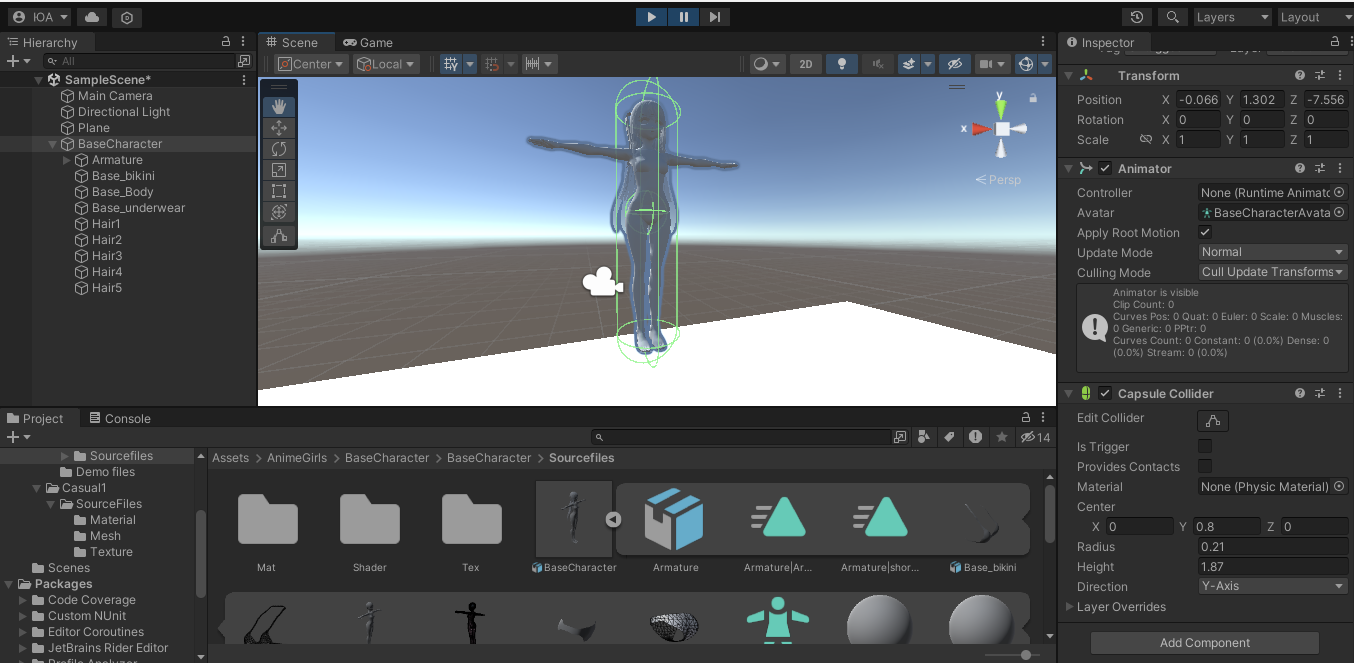


Рисунок Добавление Capsule Collider

Rigibody в свою очередь добавляет реальную физику к объекту, что позволяет ему реагировать на гравитацию, силы, столкновения и т.д. Rigidbody делает объект динамическим, позволяя ему двигаться, падать и реагировать на силы в игровом мире.

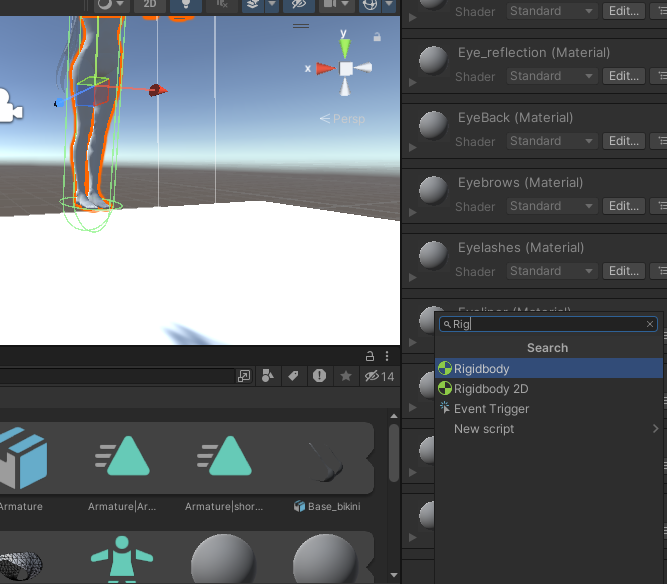


Рисунок Добавление Rigibody

Так же в Rigibody мы выбираем во вкладке Constraints Freeze Rotation для того что бы они не вращались по x и z.

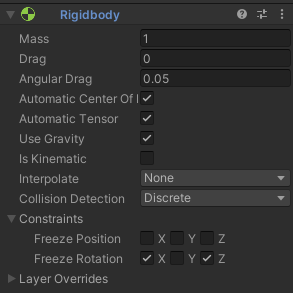


Рисунок настройка Rigibody

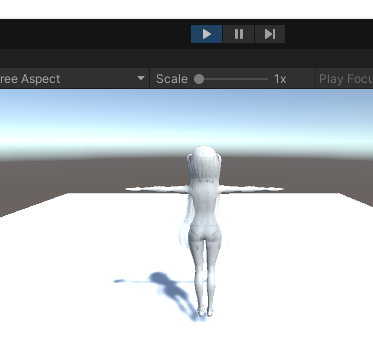


Рисунок падение модели

Теперь же мы создадим папку и сам скрипт для этого нам надо нажать правой кнопкой мыши выбрать “Создать” и выбрать папку, и аналогично надо сделать с файлом для скрипта.

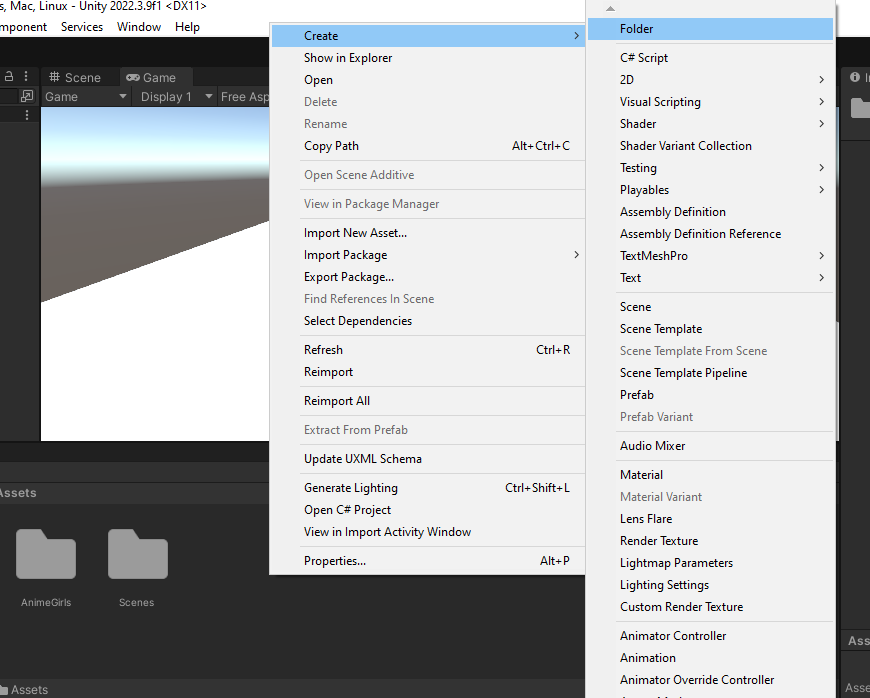


Рисунок создание папки и скрипта

Теперь же, нам надо написать код для управления этой модели. Сейчас мы должны понять за что отвечает отдельная часть кода.

* Объявление переменных: В начале кода объявляются переменные для управления персонажем. Эти переменные включают в себя скорость движения (MovementSpeed), силу прыжка (JumpForce), и расстояние до поверхности земли (DistationToGround). Также есть переменные для отслеживания состояния персонажа, такие как IsGrounded, которая указывает, находится ли персонаж на земле, и \_Rigidbody, который представляет собой физическое тело персонажа.
* Метод Start: В этом методе инициализируется \_Rigidbody, получая ссылку на компонент Rigidbody, присоединенный к этому объекту.
* Метод FixedUpdate: Этот метод вызывается на каждом кадре физики. В нем происходит следующее:
* GroundCheck проверяет, находится ли персонаж на земле, выполняя лучевой луч от нижней части персонажа вниз.
* Если клавиша пробела (Space) нажата, вызывается метод Jump, что приводит к применению силы вверх к персонажу.
* Затем вызывается метод CalculateMovement, чтобы рассчитать новую позицию персонажа.
* Новая позиция персонажа применяется с использованием \_Rigidbody.MovePosition, что обеспечивает плавное перемещение персонажа.
* Метод CalculateMovement: Этот метод определяет, как будет двигаться персонаж в ответ на ввод от игрока. Он использует значения осей горизонтали и вертикали (Horizontal и Vertical), чтобы создать вектор движения, который зависит от направления взгляда персонажа. Затем этот вектор перемножается на скорость перемещения и масштабируется на время между кадрами.
* Метод Jump: В этом методе применяется сила вверх через \_Rigidbody.AddForce, что позволяет персонажу совершить прыжок. Однако прыжок возможен только в том случае, если IsGrounded равно true, что гарантирует, что персонаж не может выполнять бесконечные прыжки в воздухе.
* Метод GroundCheck: Этот метод использует лучевой луч, который выпускается из нижней части персонажа вниз, чтобы проверить, находится ли персонаж на земле. Если луч пересекает поверхность, и расстояние до нее меньше DistationToGround, то IsGrounded устанавливается в true, что означает, что персонаж находится на земле.
* Метод OnDrawGizmosSelected: Этот метод используется для визуализации луча земли, который выпускается при проверке на землю. Он рисует красную линию от позиции персонажа вниз на расстояние DistationToGround.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerControler : MonoBehaviour

{

public float Sensivity = 10f;

public float RotationSmoothing = 20f;

public float MovementSpeed = 2.0f;

public float JumpForce = 5.0f;

public float DistationToGround = 0.1f;

private float pitch;

private float yaw;

private bool IsGrounded;

private Rigidbody \_Rigidbody;

public GameObject HandMeshes;

public float SprintSpeed = 10.0f;

private GameManager \_GameManager;

void Start()

{

\_Rigidbody = GetComponent<Rigidbody>();

\_GameManager = FindObjectOfType<GameManager>();

}

// Update is called once per frame

void FixedUpdate()

{

GroundCheck();

if (Input.GetKey(KeyCode.Space)&& IsGrounded) Jump() ;

\_Rigidbody.MovePosition(CalculateMovement());

if (Input.GetKey(KeyCode.LeftShift) && !\_GameManager.IsStaminaRestoring)

{

\_GameManager.SpendStamina();

\_Rigidbody.MovePosition(CalculateSprint());

}

else \_Rigidbody.MovePosition(CalculateMovement());

SetRotation();

}

public void SetRotation()

{

//Считываем движения мыши по X

yaw += Input.GetAxis("Mouse X") \* Sensivity;

//Считываем движения мыши по Y

pitch -= Input.GetAxis("Mouse Y") \* Sensivity;

//Делаем ограничение вращения по оси Y

pitch = Mathf.Clamp(pitch, -60, 90);

//Куда должен повернуться игровк

Quaternion SmoothRotation = Quaternion.Euler(pitch, yaw, 0);

//Линейная интерполяция (вращение со сглаживанием)

HandMeshes.transform.rotation = Quaternion.Slerp(HandMeshes.transform.rotation, SmoothRotation, RotationSmoothing \* Time.fixedDeltaTime);

//То же самое для всего игрового объекта, но только по оси Y

SmoothRotation = Quaternion.Euler(0, yaw, 0);

transform.rotation = Quaternion.Slerp(transform.rotation, SmoothRotation, RotationSmoothing \* Time.fixedDeltaTime);

}

private Vector3 CalculateMovement()

{

float HorizontalDirection = Input.GetAxis("Horizontal");

float VerticalDirection = Input.GetAxis("Vertical");

Vector3 Move = transform.right \* HorizontalDirection + transform.forward \* VerticalDirection;

return \_Rigidbody.transform.position + Move \* Time.fixedDeltaTime \* MovementSpeed;

}

private Vector3 CalculateSprint()

{

float HorizontalDirection = Input.GetAxis("Horizontal");

float VerticalDirection = Input.GetAxis("Vertical");

Vector3 Move = transform.right \* HorizontalDirection + transform.forward \* VerticalDirection;

return \_Rigidbody.transform.position + Move \* Time.fixedDeltaTime \* /\* Меняется только это -> \*/ SprintSpeed;

}

private void Jump()

{

\_Rigidbody.AddForce(Vector3.up \* JumpForce, ForceMode.Impulse);

}

private void GroundCheck()

{

IsGrounded = Physics.Raycast(transform.position, Vector3.down, DistationToGround);

}

private void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawLine(transform.position, transform.position + (Vector3.down \* DistationToGround));

}

}

Далее после того, как мы написали скрипт, нам надо привязать его к объекту. Для этого мы берём скрипт и перетаскиваем туда, где у нас хранятся другие 2 компонента.

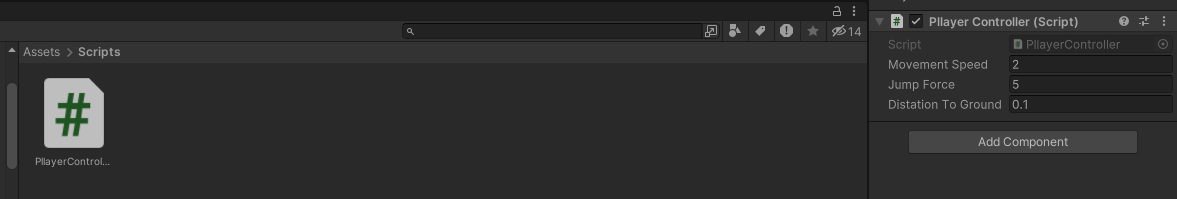


Рисунок привязка скрипта

Теперь наша модель может перемещаться и прыгать на сцене.

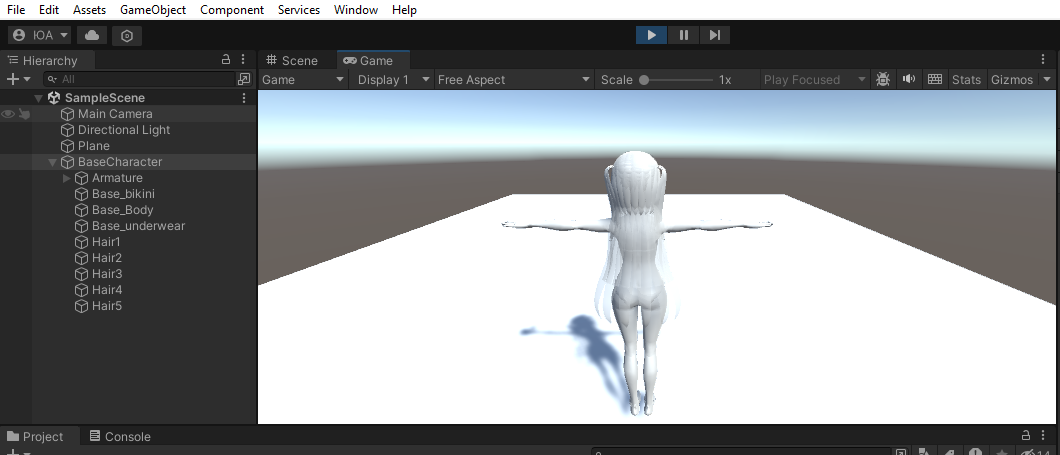


Рисунок результат

Вывод: При помощи базовых знаний интерфейса Unity и языка программирования С#, реализовали передвижение персонажа.