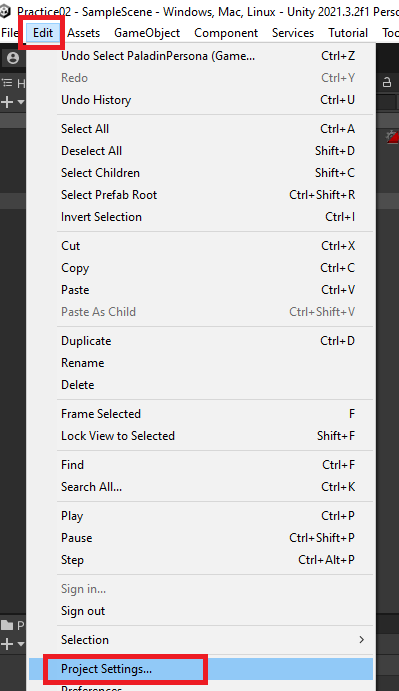
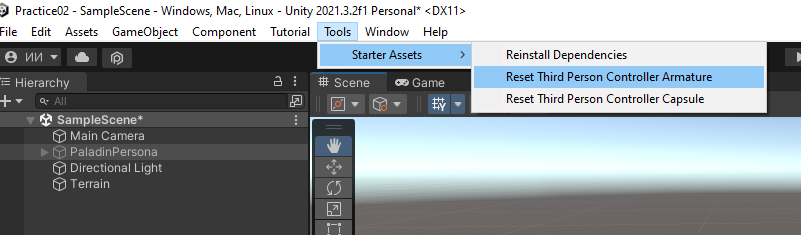
После установки Starter Assets

# Переключение методов ввода управления персом старый/новый

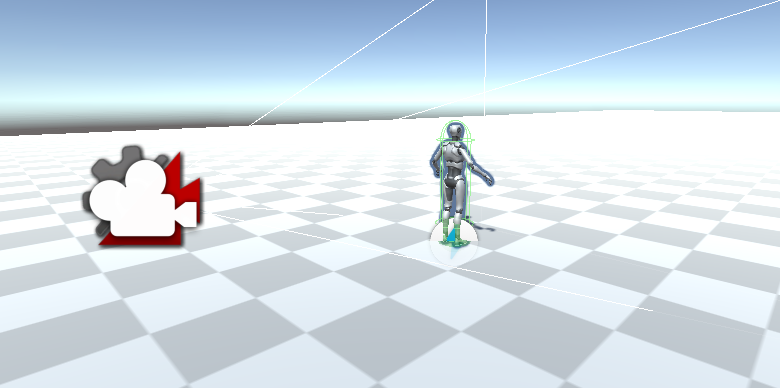




# Установка стандартного перса из Starter Assets



Находим персонажа на карте – он может быть под террейном или хз где и ставим его на карту:

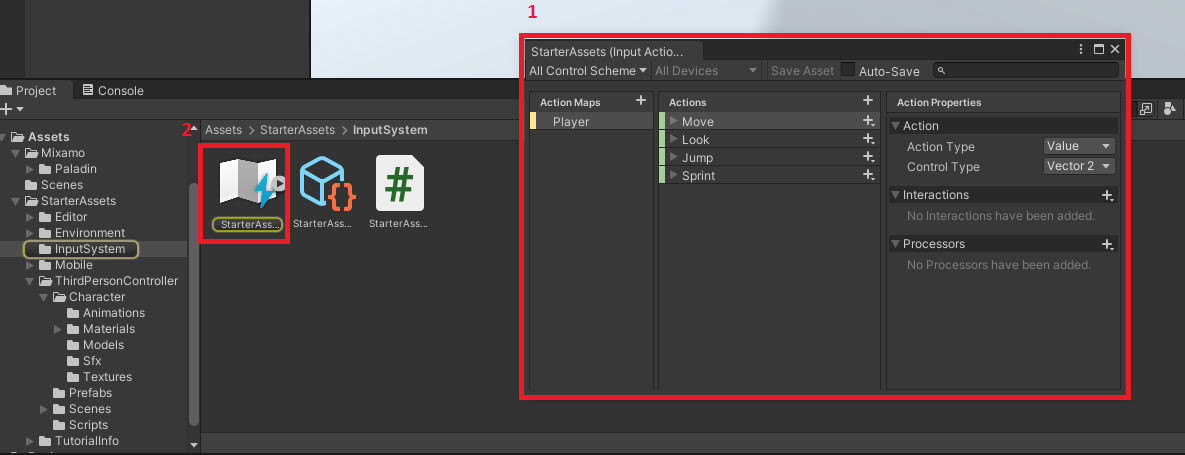


Этот персонаж сразу может уже ходить, бегать и прыгать.

Находим у этого перса в инспекторе скрипт Player Input и кликаем по:



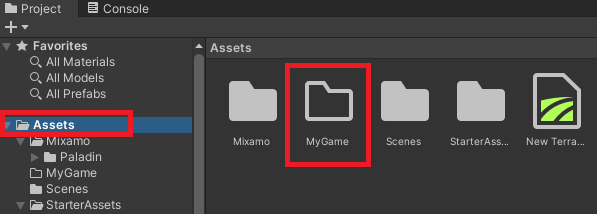
Видим:



1 - Появляется окно с настройкой клавиш ввода

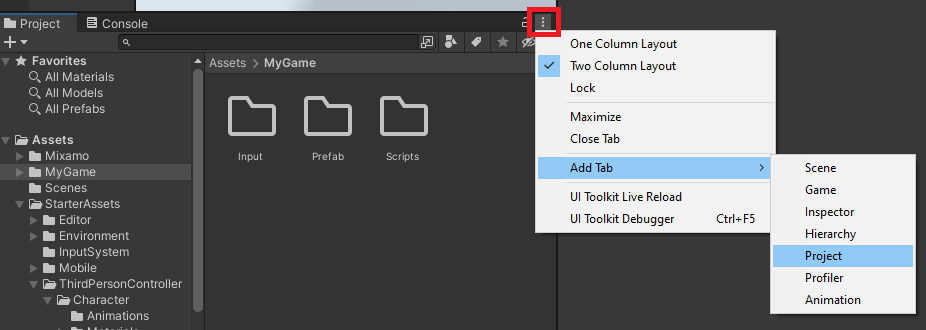
2 – файл который обрабатывает ввод с клавиш. Нам нужно отвязать этот ввод от стандартного персонажа что бы потом привязать его к своему персонажу. По этому этот файл нужно будет забрать отсюда в папку MyGame.

Создадим в ассетах свою папку (MyGame) :

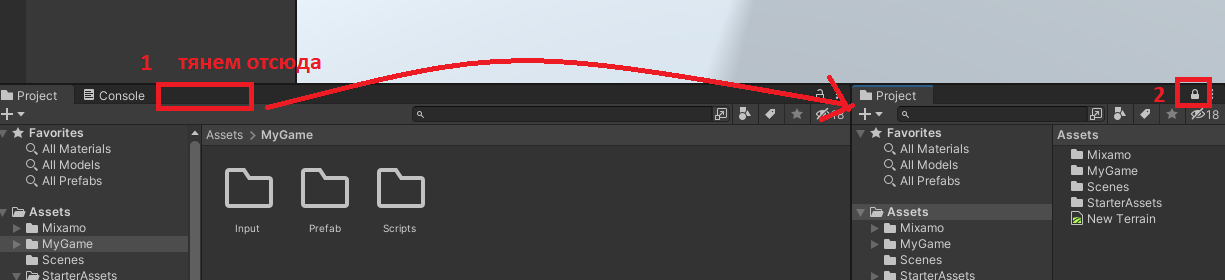


# Делаем 2 вкладки со структурой проекта (вторая для моей личной папки что бы ее не искать все время)

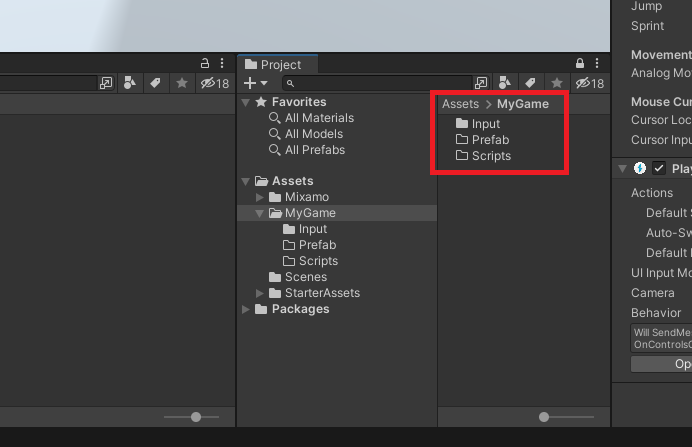
Разделяем вкладки с проектами:



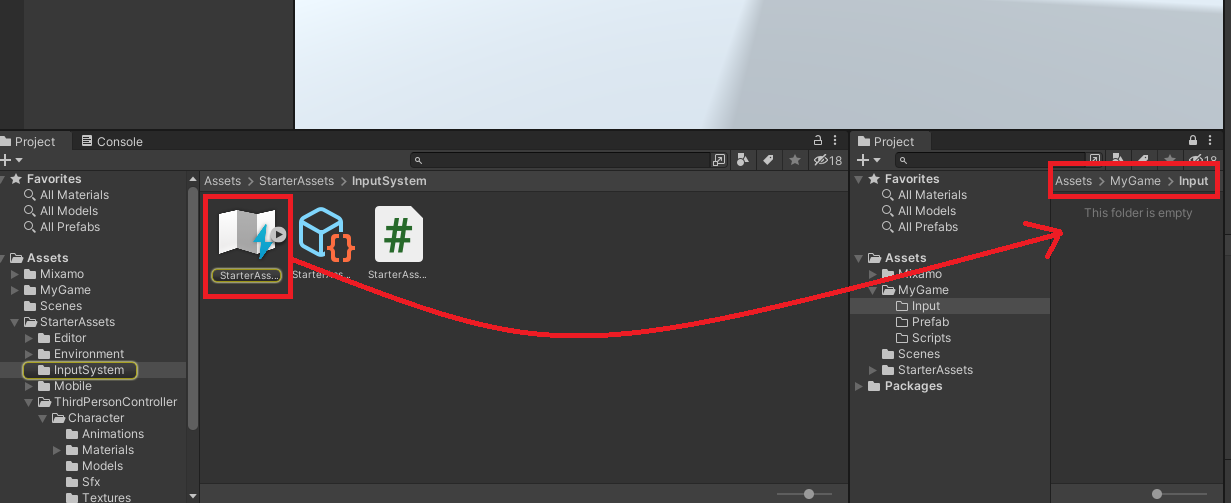
Появившуюся вкладку тянем вправо и закрепляем:



Выберем во второй вкладке структуры проекта нашу папку MyGame и создадим внутри нее еще 3 папки:

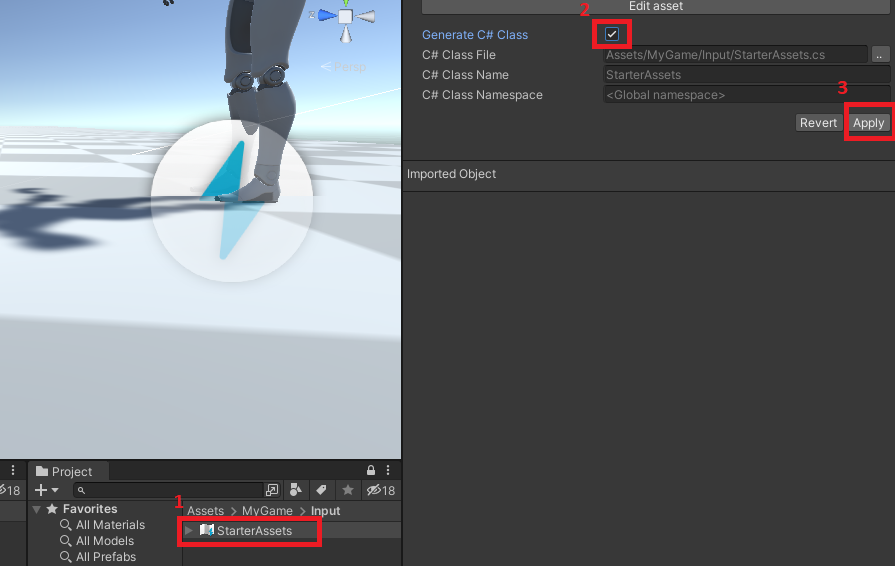


Теперь тот файл, который мы дорлжны забрать переместим (перетащим мышкой) в папку Input:



Переменуем Этот файл в MyInputSystem для того что бы удобно находить его и понимать что это наш переделанный файл

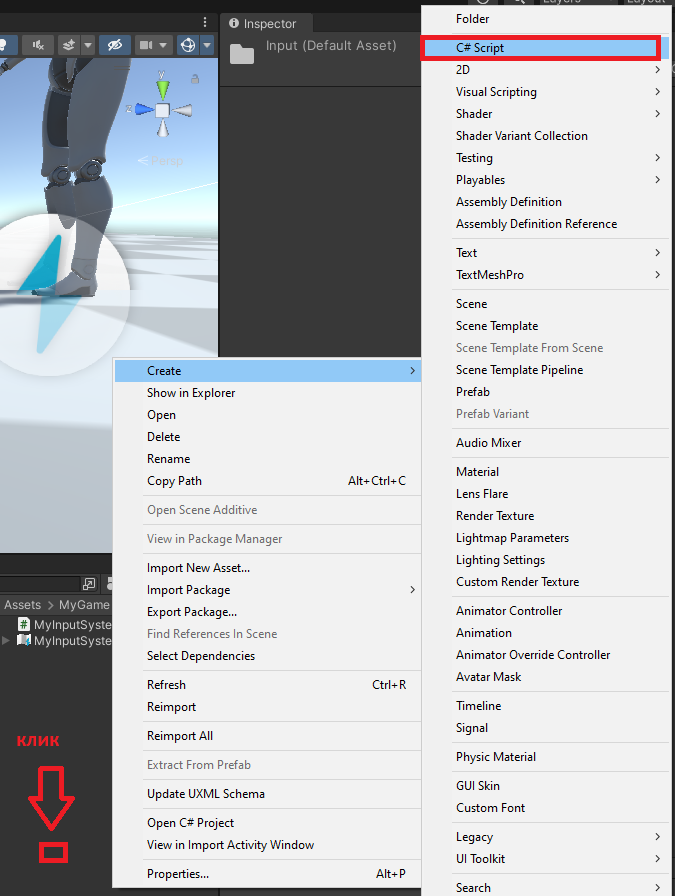
Теперь выделим этот файл (один раз кликнув на него) и сгенерируем его си шарп код:



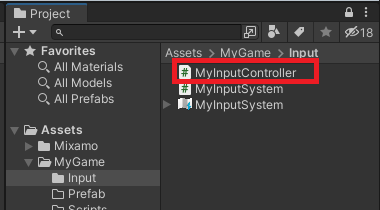
Название этого файла будет уже MyInputSystem

Теперь мы отвязали этот код от стандартного персонажа

И создадим здеь же еще один скрипт:



Получм:



Этот файл будет отвечать за ввод пользователя – то есть в Character controller которым мы будем управлять мы будет получать все данные УЖЕ ИЗ ЭТОГО КЛАССА.

Открываем этот файл в слудии Переименовываем название класса в MyInputController (как называется этот файл) и вставляем этот код:

public class MyInputController : MonoBehaviour // название класса переименовали

{

// MyInputSystem - название файла, который мы взяли из Starter Assets и переименовали

MyInputSystem input;

public Vector2 move;

public Vector2 look;

// переменные состояний (для проверки в каком положении находимся)

public bool sprint;

public bool crouch;

public bool jump;

public bool hit;

// геттеры - т.е. можем только получать переменные. Будем обращаться именно к ним (что бы случайно не поменять значения)

public Vector2 Move => move;

public Vector2 Look => look;

public bool isSprint => sprint;

public bool isCrouch => crouch;

public bool isJump => jump;

private void Awake()

{

input = new MyInputSystem();

// move - W, A, S, D + arrows

input.Player.Move.started += OnMove; // started - когда начали нажимать

input.Player.Move.performed += OnMove; // performed - держим клавишу зажатой

input.Player.Move.canceled += OnMove; // canceled - отпустили клавишу

// приставку On добавляем обязательно. Move, Look, Sprint, Jump - названия действий как мы их назвати в том окошке, где выставляем нажатие клавиш (файл с названием MyInputSystem)

input.Player.Look.started += OnLook;

input.Player.Look.performed += OnLook;

input.Player.Look.canceled += OnLook;

// здесь, например, performed не надо прописывать потому что переменная не меняется (тип переменной bool а не Vector2)

input.Player.Sprint.started += OnSprint;

input.Player.Sprint.canceled += OnSprint;

input.Player.Jump.started += OnJump;

input.Player.Jump.canceled += OnJump;

}

private void OnJump(InputAction.CallbackContext obj)

{

// ReadValueAsButton() - получение тру/фолс - нажата/ненажата клавиша

jump = obj.ReadValueAsButton();

// получается что состояние тру это только тогда, когда нажата клавиша отвечающея за прыжок

}

private void OnSprint(InputAction.CallbackContext obj)

{

sprint = obj.ReadValueAsButton();

}

private void OnLook(InputAction.CallbackContext obj)

{

// когда получаем вектор то используем ReadValue

look = obj.ReadValue<Vector2>();

}

private void OnMove(InputAction.CallbackContext obj)

{

move = obj.ReadValue<Vector2>();

}

// деактивируем MyInputSystem input;

private void OnDisable()

{

input.Player.Disable();

}

// активируем MyInputSystem input;

private void OnEnable()

{

input.Player.Enable();

}

}

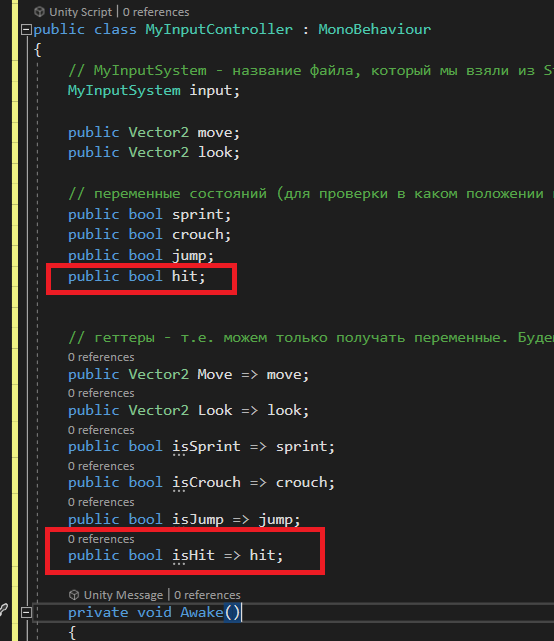
# Добавляем к управлению персонажем действие по нажатию какой-то кнопки

Добавим действие удара:

1) добавляем переменную и геттер:

public bool hit;

public bool isHit => hit;



2) добавляем метод который вызывается по действию удара с подпискойна него:

input.Player.Hit.started += OnHit;

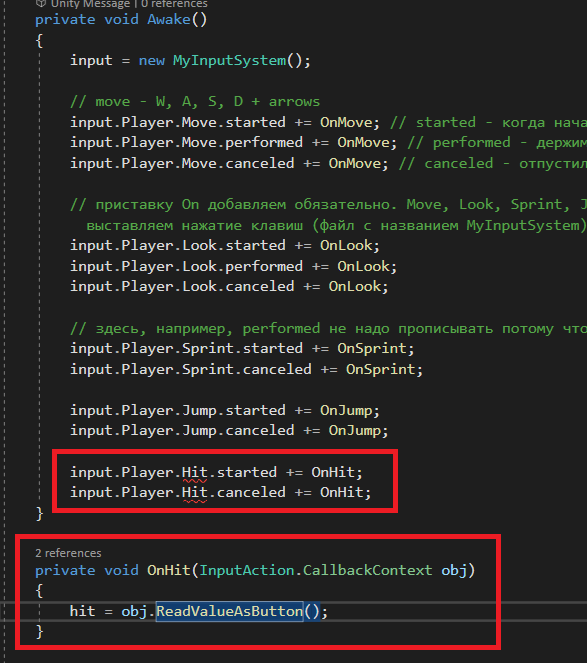
input.Player.Hit.canceled += OnHit;

private void OnHit(InputAction.CallbackContext obj)

{

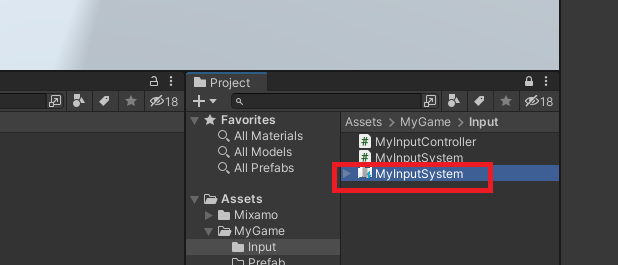
hit = obj.ReadValueAsButton();

}

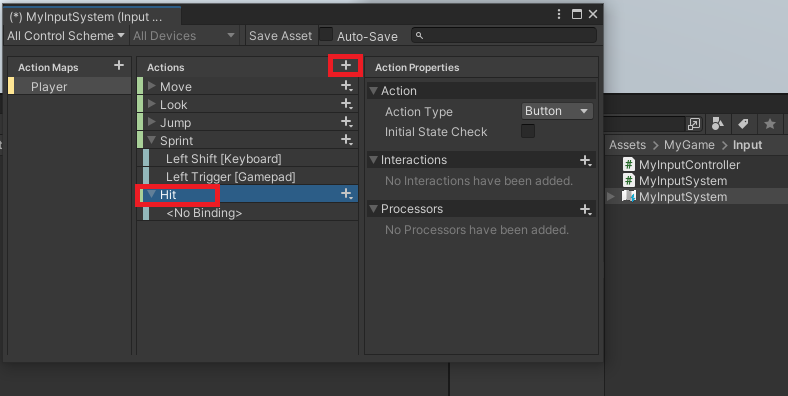


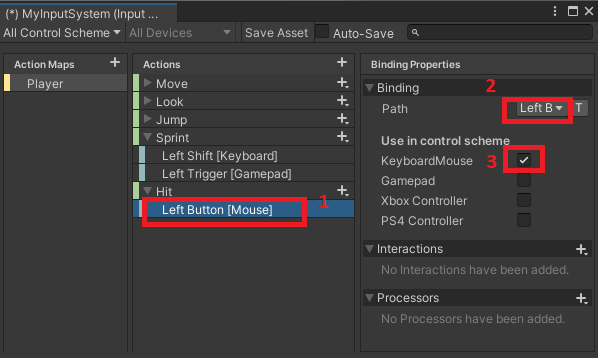
Видим что подчеркивает красным название действия, это означает что мы не назначили клавишу для этого действия. Назначим:

Откроем этот файл:



Add new action:



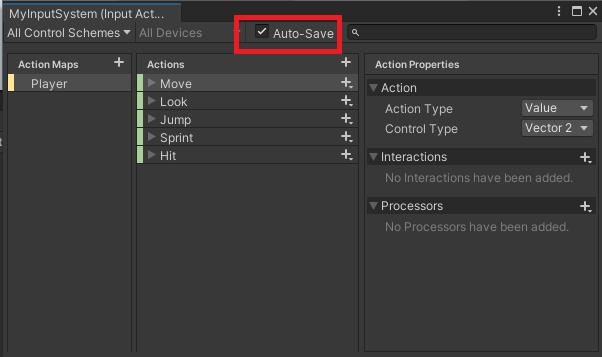


1 – выбираем привязку

2 – выбираем ЛКМ

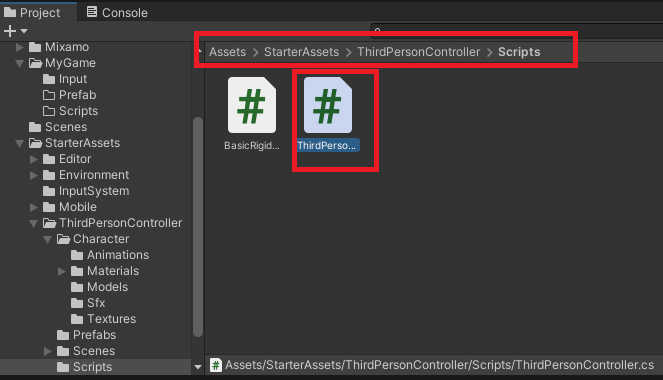
3 – говорим о том что это мышка или клавиатура

И поставим галочку на авто сохранении:

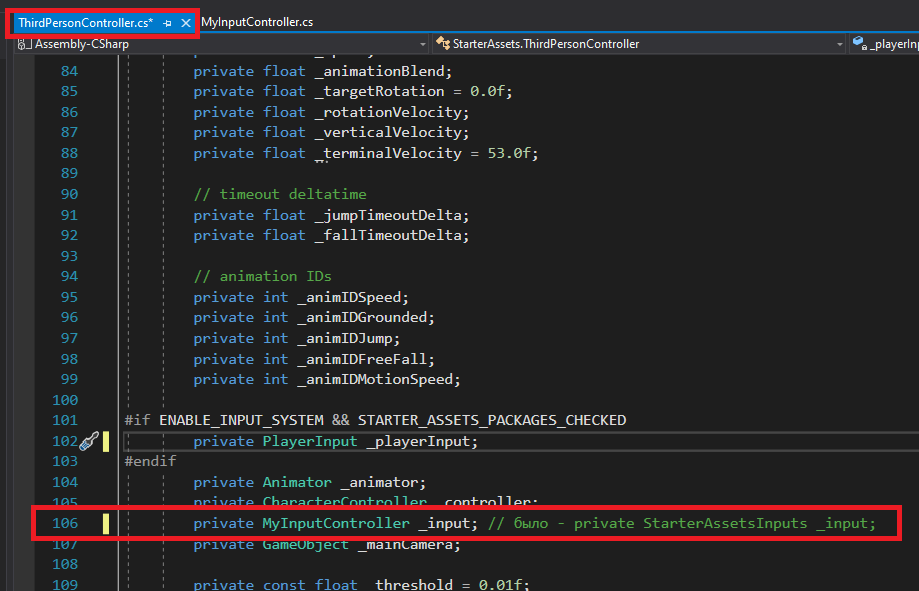


Теперь ошибка пропадет

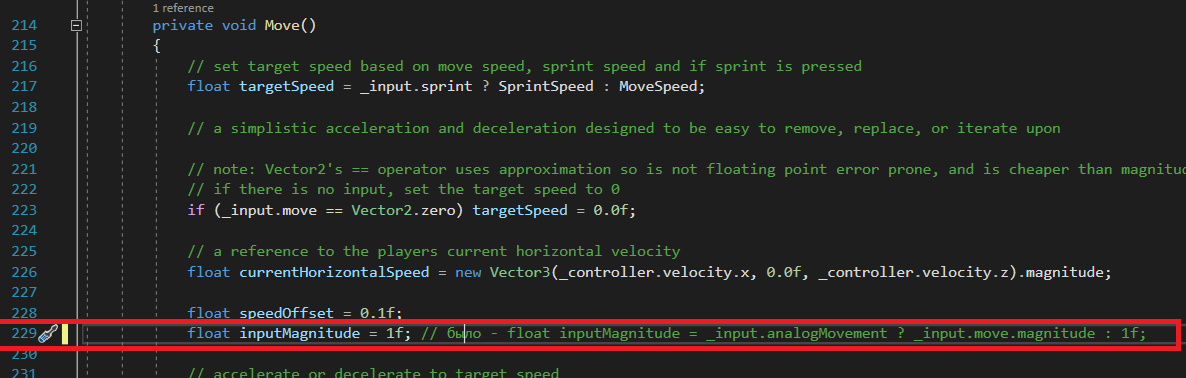
Теперь Меняем следующий скрипт – находим скрипт из стартер ассетс:



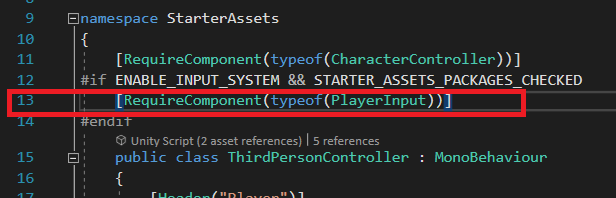
Открываем его в студии и меняем имя на наш только что созданный класс:



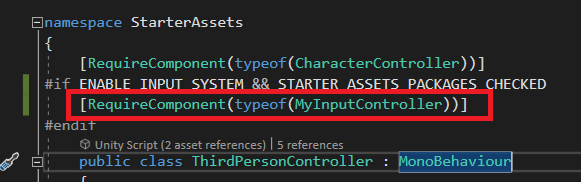
Так же ищем по ошибкам где еще был тот класс и меняем его на наш



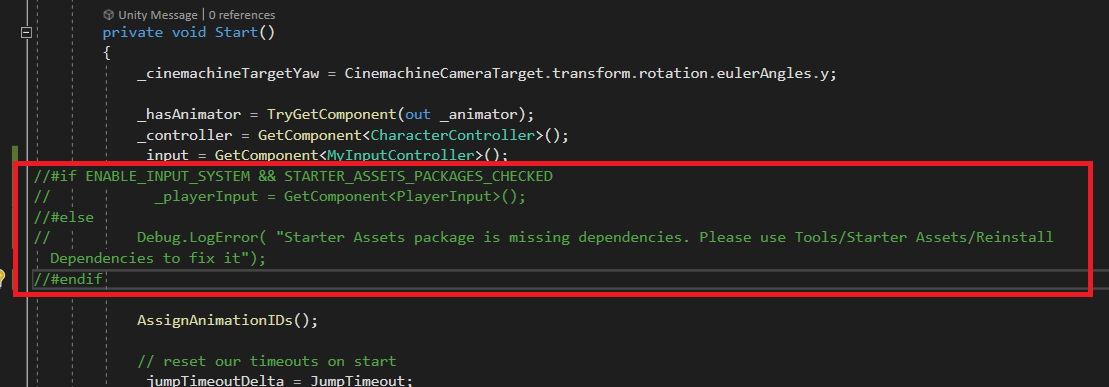
Удаляем строку:

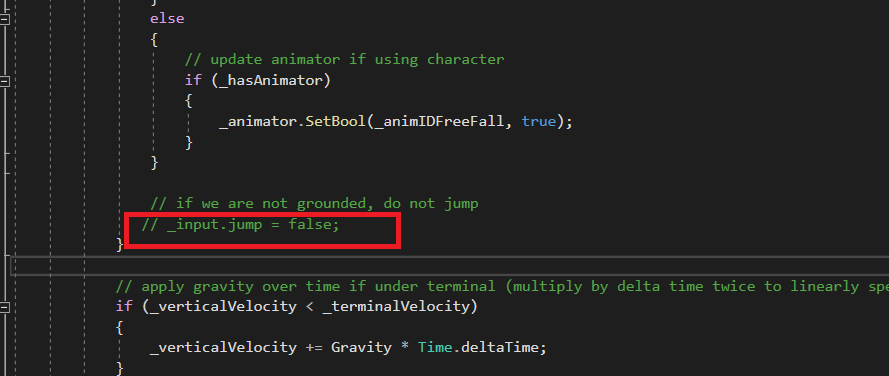


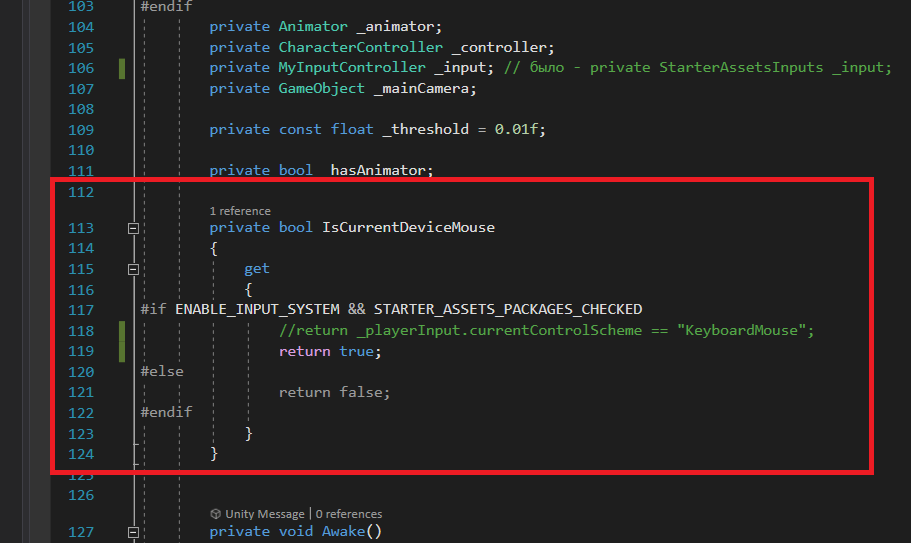
Вставляем вместо нее:



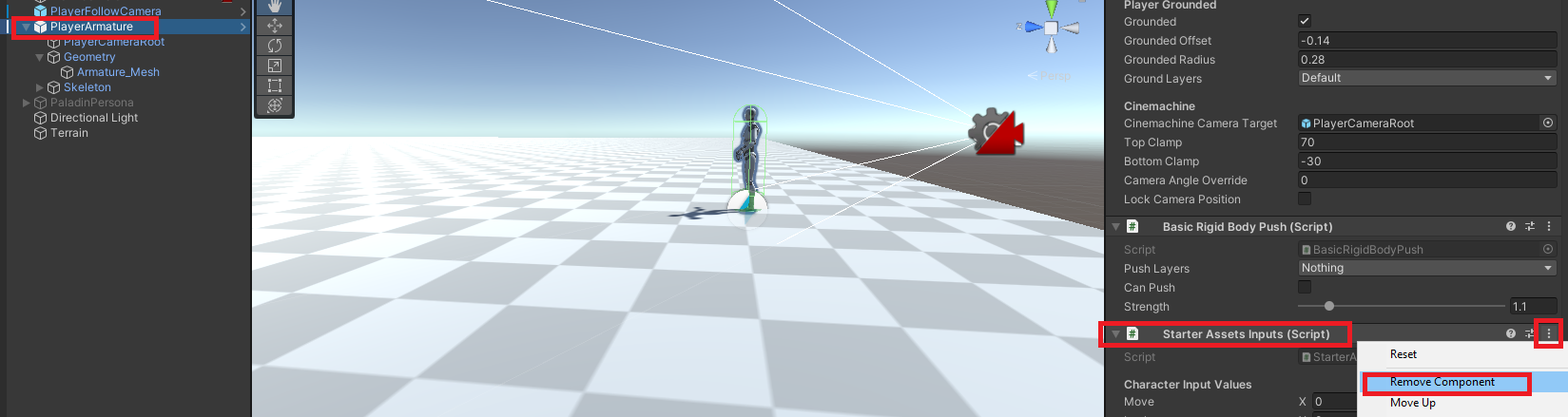
А так же нужно закомментить (удалить):



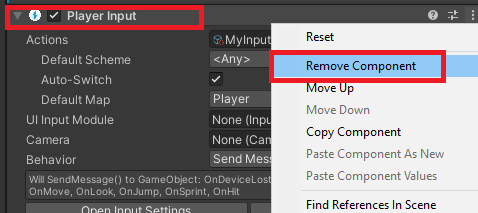




Удаляем лишние скрипты у нашего стандартного перса:



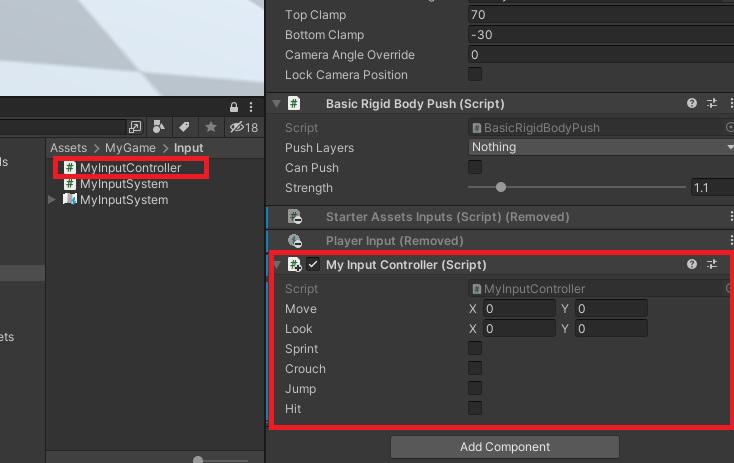
И



И вместо этих скриптов добавим персу свои:

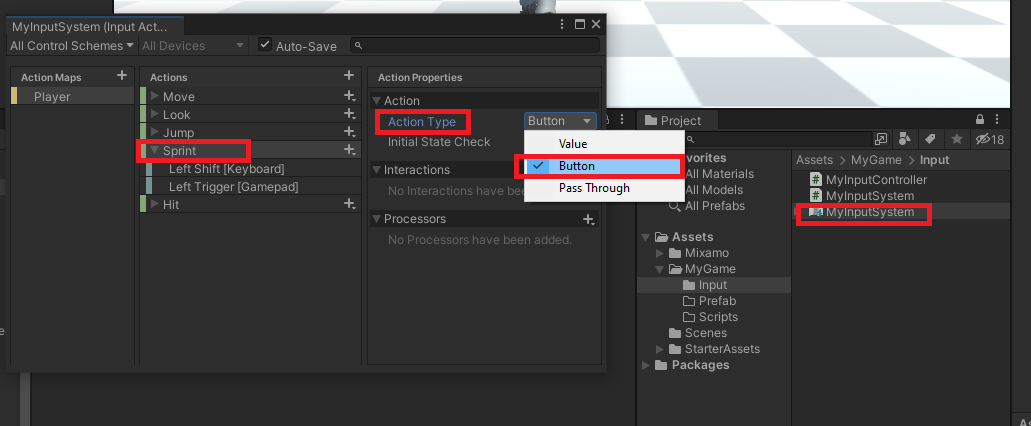


Result:

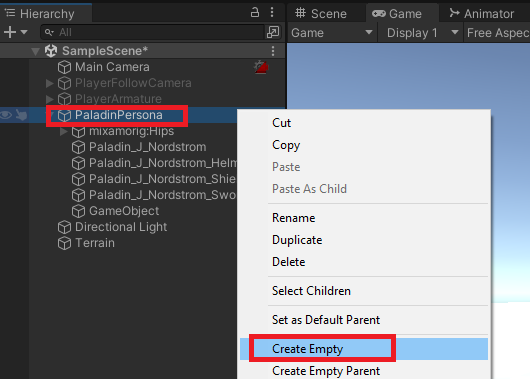


Теперь можем запускать - все должно работать

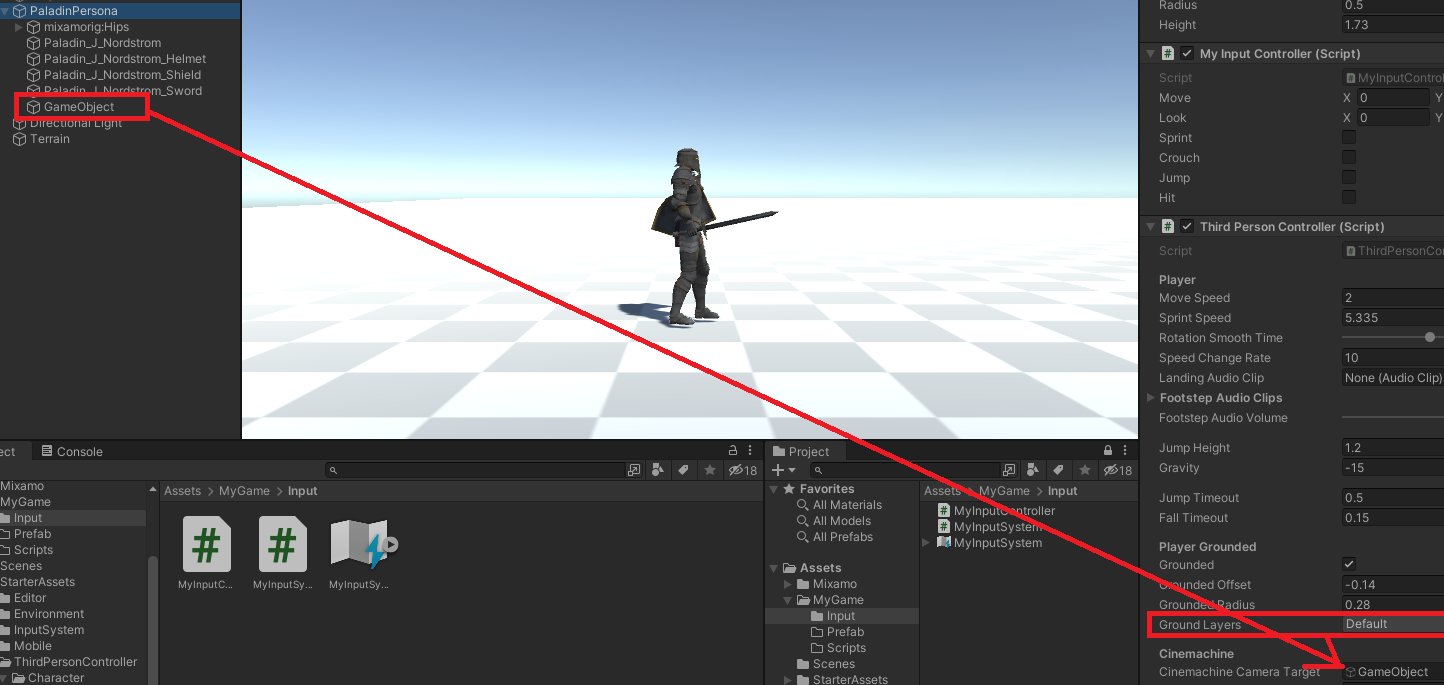
Если перс не бегает, убедимся что тип нажатия кнопки при беге именно «кнопка»:

****

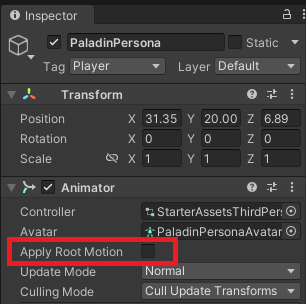
# Ставим все управление на перса с михамо



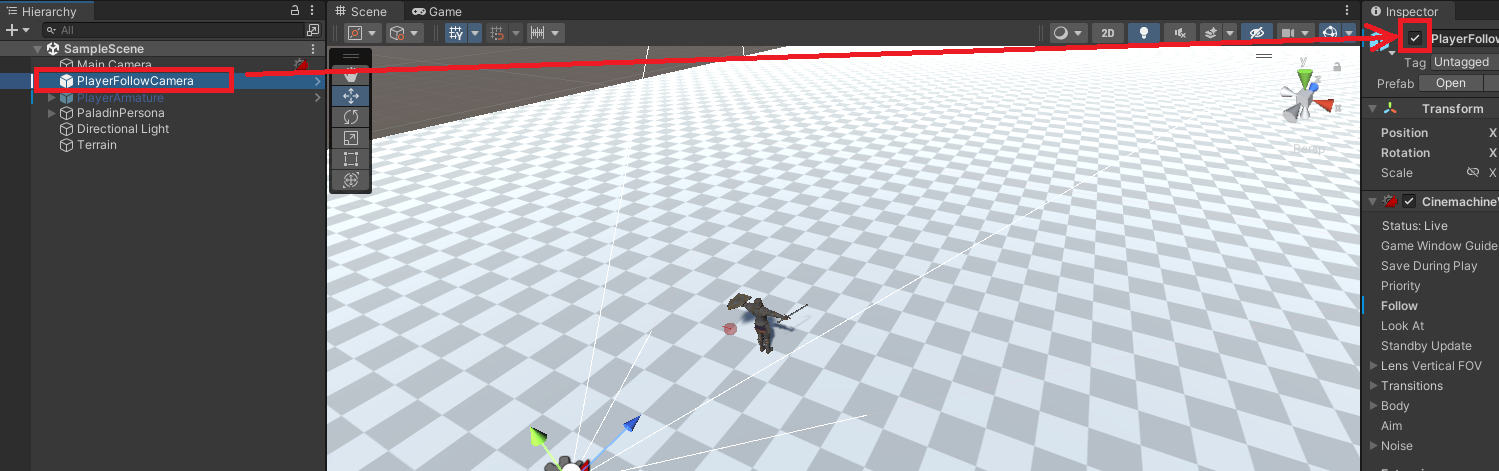
Перемещаем созданный гейм обджект и выставляем граунд леерс – дефолт:



Далее убираем галочку в аниматоре перса:



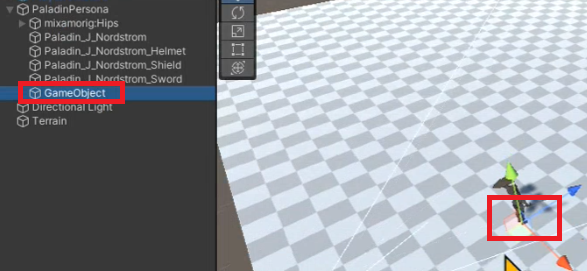
Камеру будем использовать ту что была у стандартного перса (она должна быть активна на экране):



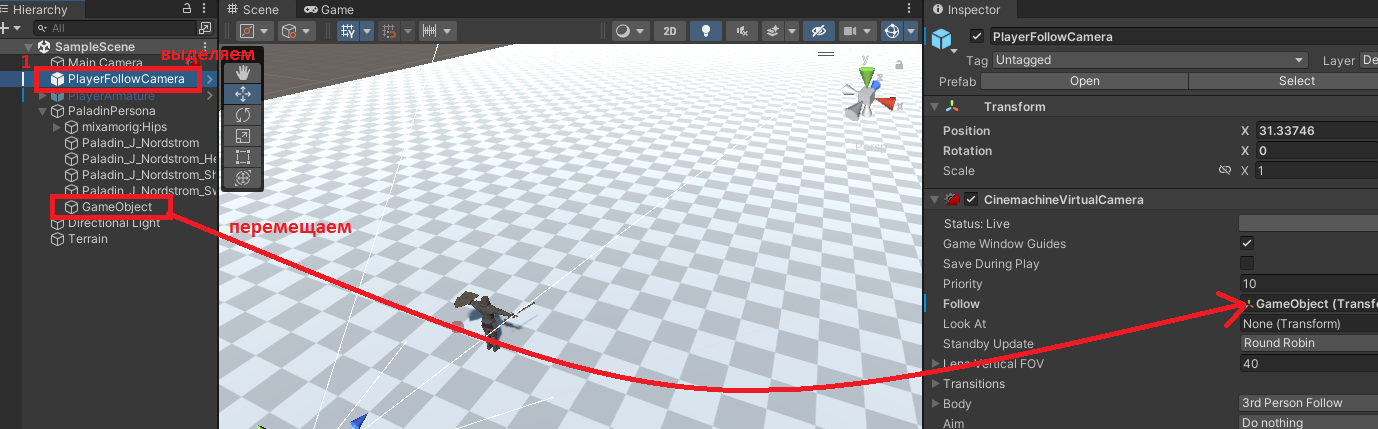
То есть стандартного перса прячем, но камеру оставляем:



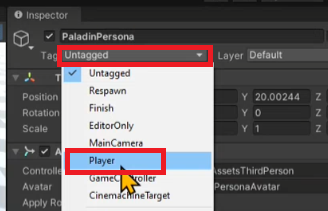
Теперь объект, который мы создали для камеры нужно выделить и выставить (объкт со стрелочками на карте) за персом сзади на уровне груди примерно:



И перетянем сюда:

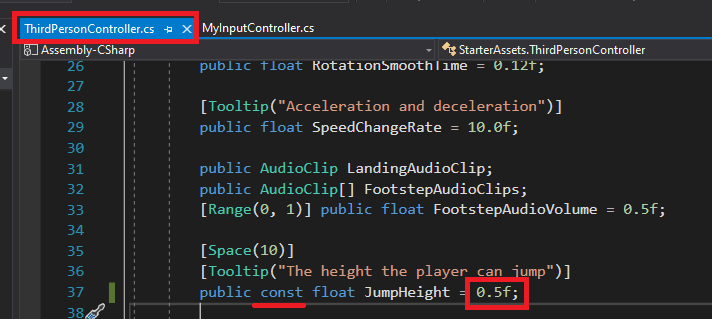


А так же у перса выставить тэг:



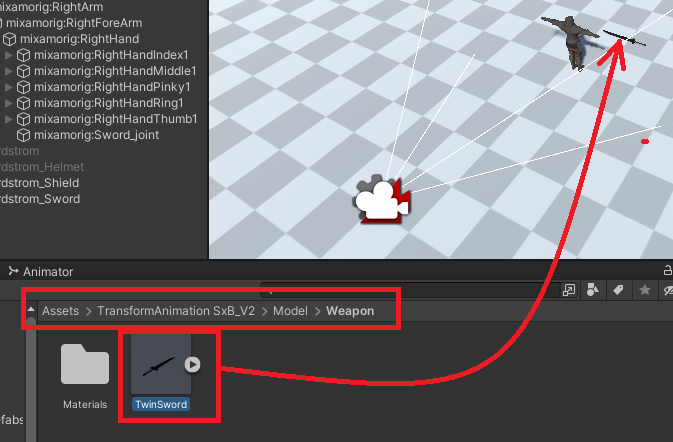
## Если проблемы с анимацией прыжка (перс застывает в воздухе)

- уменьшаем высоту прыжка в коде (делаем поле константой):

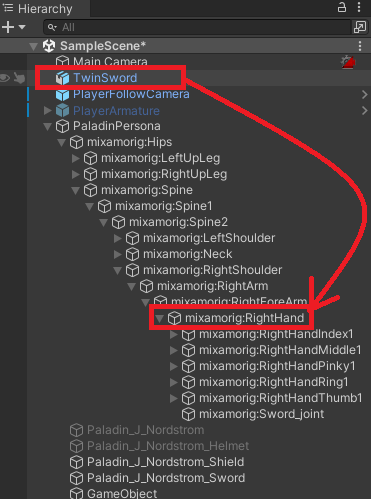


# Вкладываем сторонний меч в руку персона

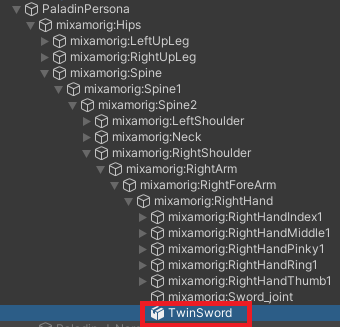
Находим меч в скачанном ассете и вытаскиваем его на карту



Вложим его в правую руку перса. Для этого развернем в иерархии объект до правой руки и с с иерархии, где появился этот меч перетянем его в правую руку:

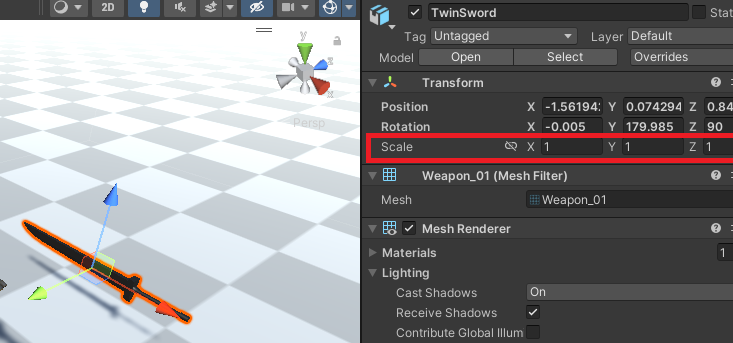


Получим:

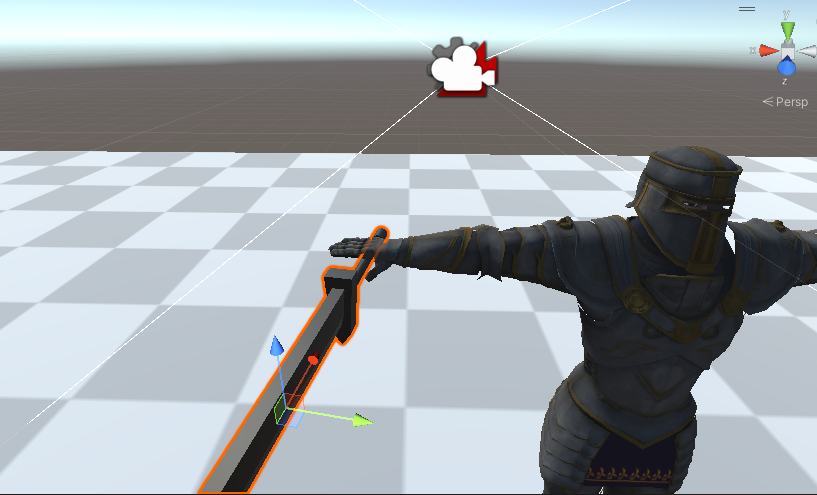


Теперь на карте нужно сам меч подогнать по размерам и вставить персу в правую руку:

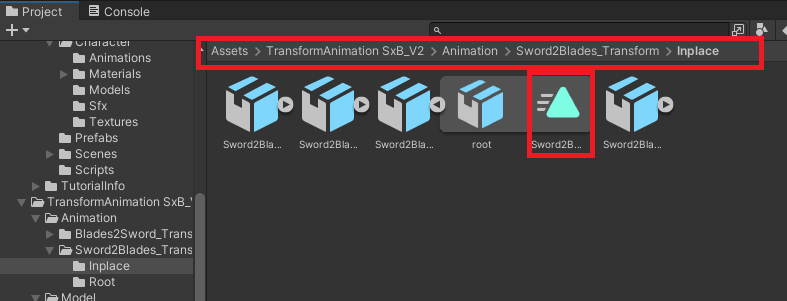
Размер:



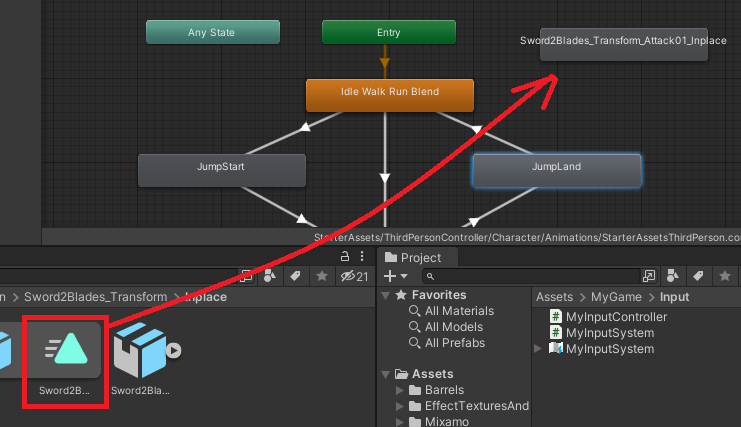
Теперь развернем его и вставим примерно так:



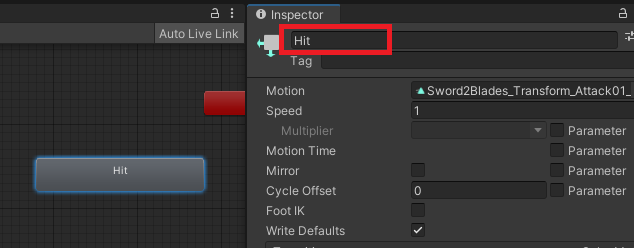
Дальше найдем анимацию удара в нашем скачанном ассете:



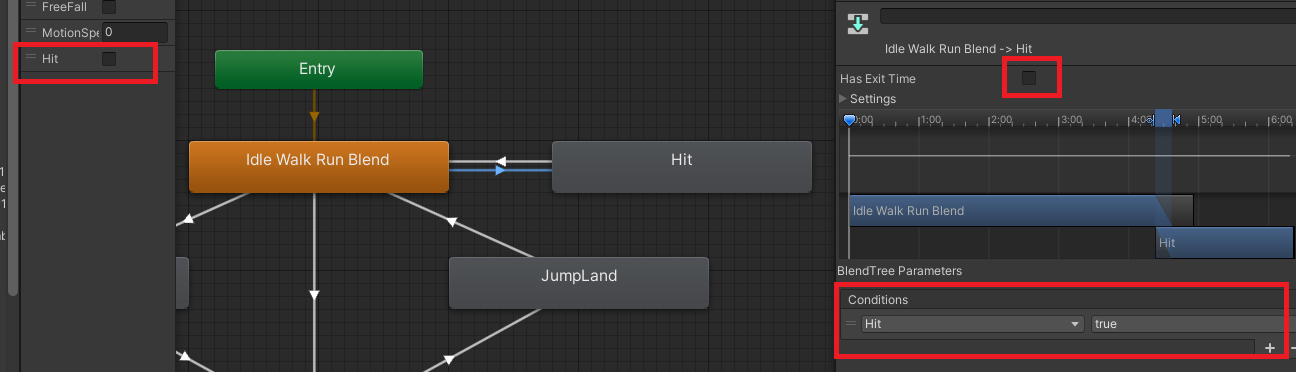
И вытянем его в аниматор персонажа:



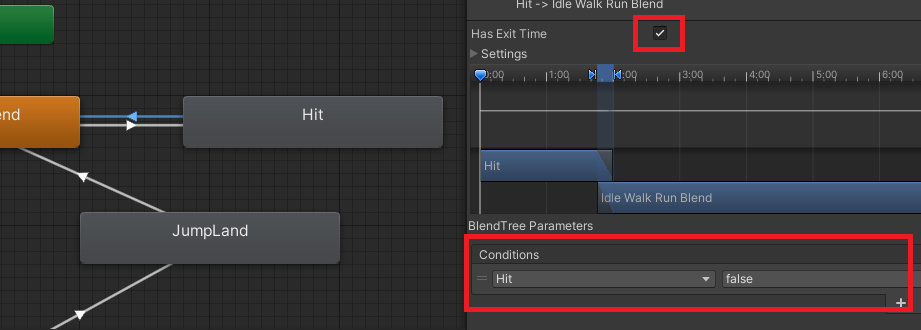
Переименуем ее:



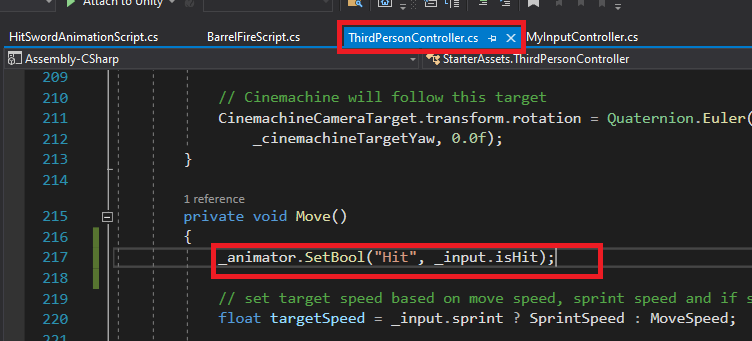
Добавим bool параметр и создадим связь туда:



И обратная связь:

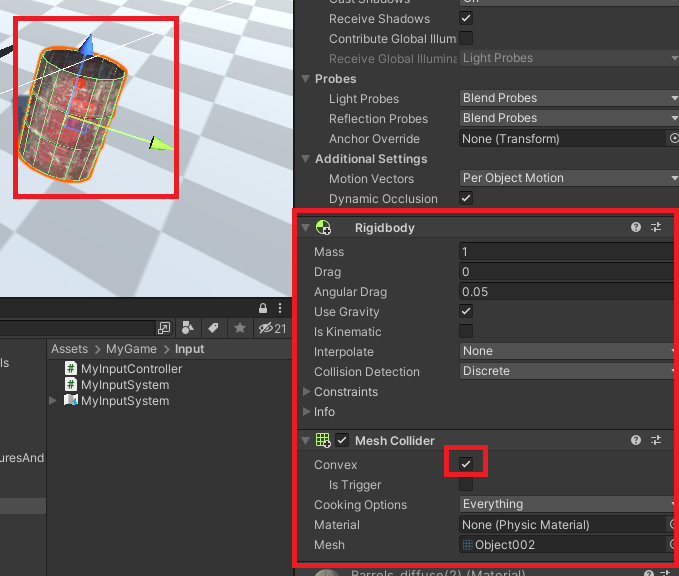


И добавим строку кода в скрипт перса что бы анимация срабатывала:

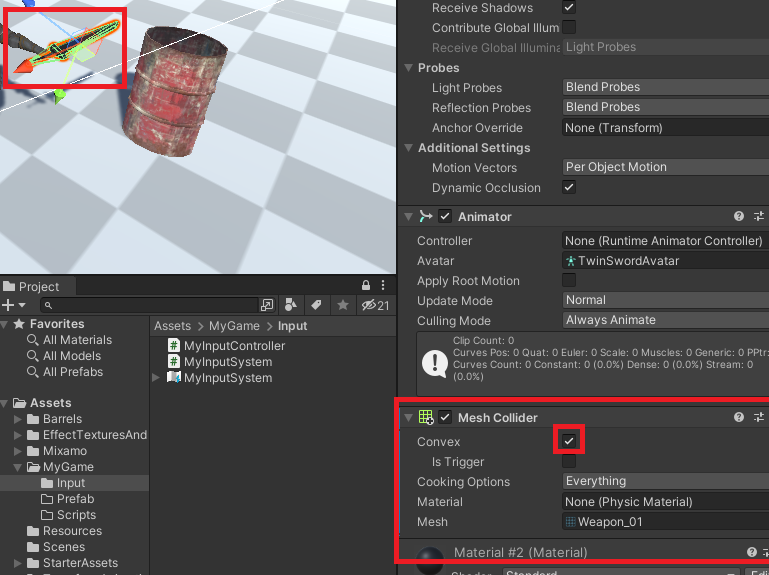


# Взаимодействие меча с бочкой

Повесим на бочку RigidBody and Mesh Collider (с галочкой):

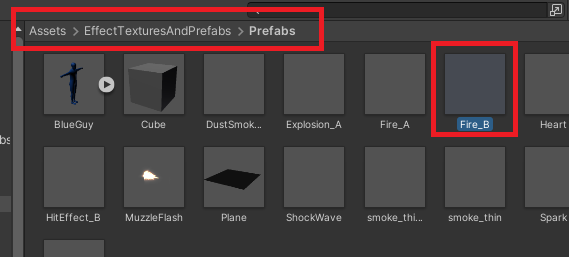


И на меч ток Mesh Collider :



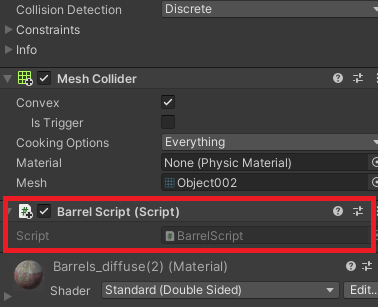
Теперь если мы только подойдем к бочке – она сразу будет падать из-за того что коллайдеры на мече и на бочке соприкосаются.

Добавим на бочку огнонек:

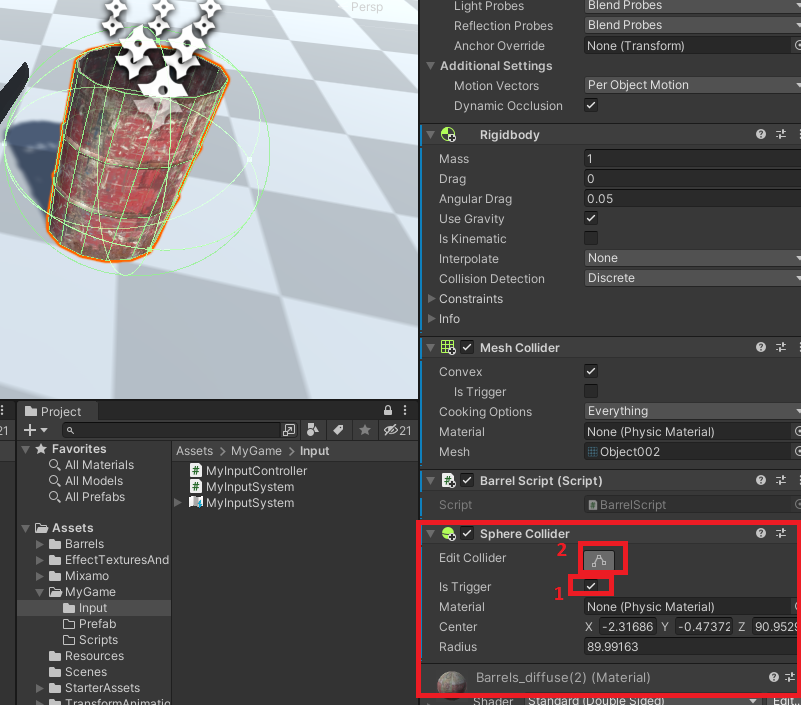




Для того что бы можно было взаимодействовать с бочкой – повесим на нее скрипт:



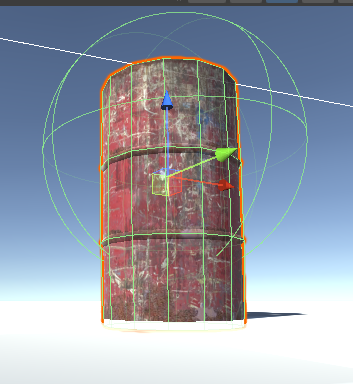
И Sphere Collider:



1 - делаем сферу тригером

2 – увеличиваем размеры (круглая зеленая сфера вокруг бочки)

Поднимаем сферу что бы она не касалась земли:



Благодаря этой сфере мы сможем отловить событие что что-то к ней приблизилось. У тригера есть 3 состояния:

- enter (что-то зашло в зону тригера – в данном случае дотронулось до сфер колайдера),

- stay (что-то находится в контакте со сферой),

- exit (что-то вышло с нее зоны действия)

Теперь откроем скрипт бочки и напишем в ней код:

public class BarrelScript : MonoBehaviour

{

public GameObject fire;

// Start is called before the first frame update

void Start()

{

}

public void OnTriggerEnter(Collider other)

{

fire.SetActive(true);

}

public void OnTriggerExit(Collider other)

{

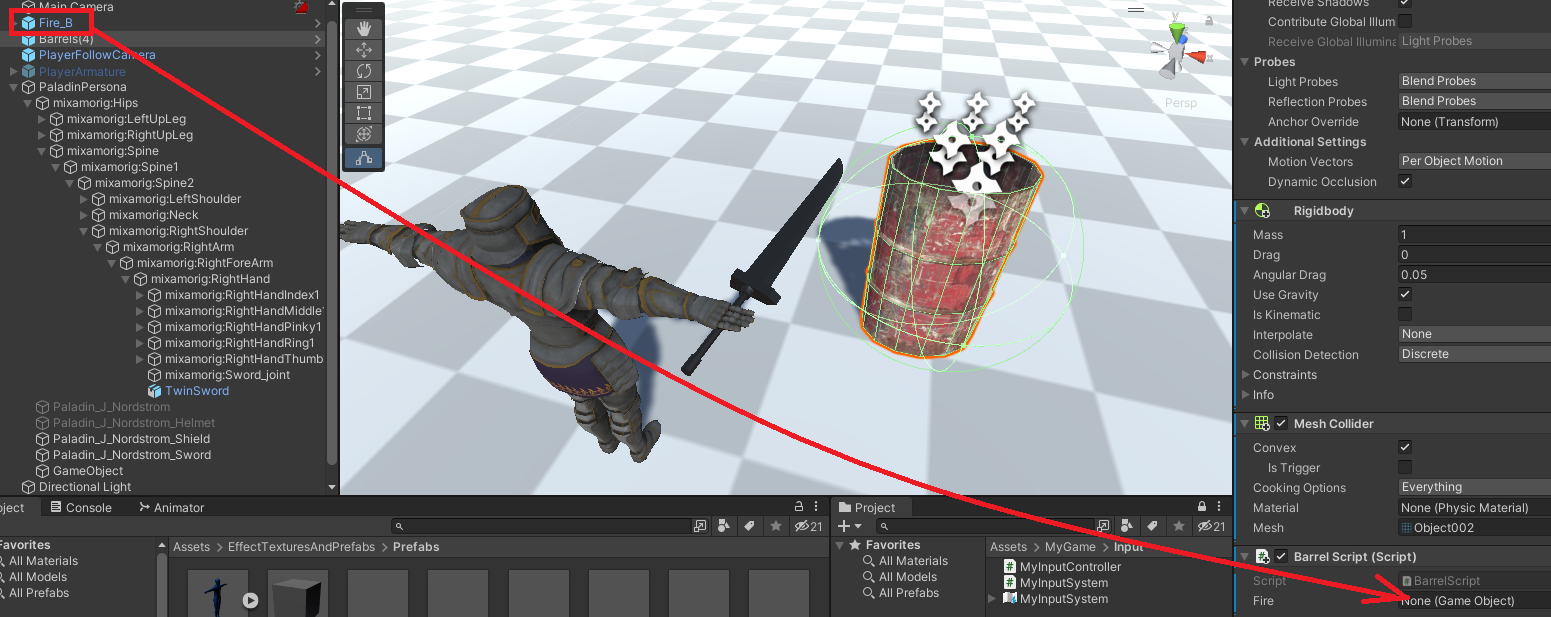
fire.SetActive(false);

}

}

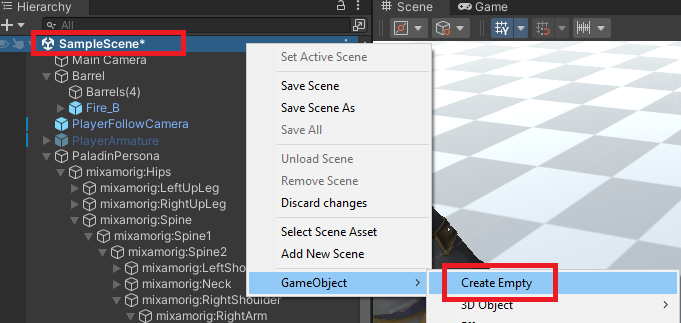
То есть при страбатываении тригера огонь будет появляться и исчезать.

Теперь нам нужно перетащить объект «огонь» в скрипт что бы в скрипте было понятно какой именно огонь имеется в виду:



Теперь при запуске игры подходим к бочке – огонь загорается, отходим – тухнет

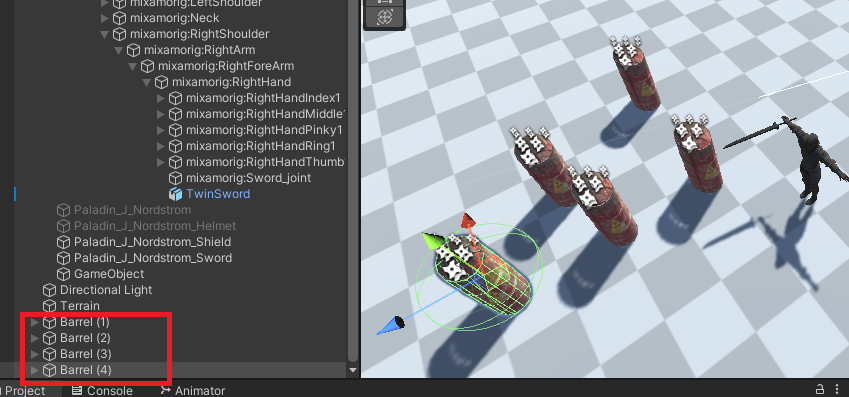
Положим бочку и огонь в один объект, который назовем Barrel , что бы можно было создать несколько таких бочек со всеми коллайдерами и скриптами и (копировать их). Для этого создаем:



Переименовываем в Barrel и перетаскиваем в этот объект нашу бочку и огонь:



Копируем этот объект и вставляем (contr+V) – там же в иерархии. Вставляем по одной, вытаскиваем бочку с того же места где стояла первая (они скопировались с одинаковыми координатами) и расставляем их по карте:



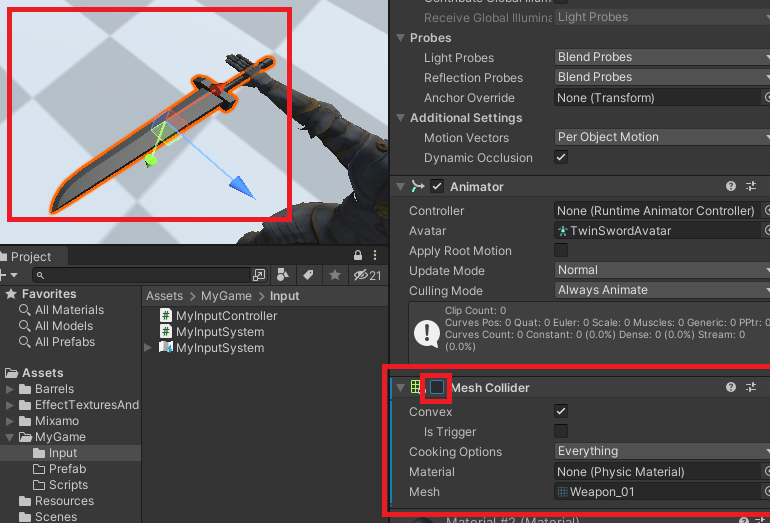
Отключаем огонь над каждой бочкой в иерархии - по умолчанию огня не будет, он загорится (станет видимым в иерархии) когда перс соприкоснется со сфер коллайдером бочки и погаснет огонь когда перс отойдет от бочки.



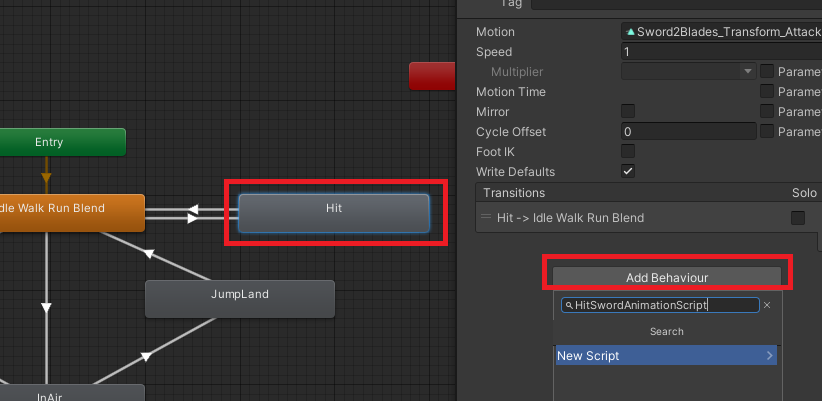
Теперь при запуске игры к какой бочке не подходи – все будут загораться, топому что имеют одинаковые настройки.

## Сделать что бы бочка не падала из-за меча, когда перс прост к ней подходит

Поскольку у бочки и у меча есть меш коллайдер то она может упать от соприкосновения с мечем, даже если не бить мечем по бочке, а просто дотронуться случайно когда перс подходит к ней. Исправим ситуацию. Сначала отключим меш коллайдер на мече:



Идем в аниматор перса, где анимация удара меча добавляем новый скрипт туда:



Открываем скрипт и пишем:

public class HitSwordAnimationScript : StateMachineBehaviour

{

public GameObject sword;

private void Awake()

{

sword = GameObject.Find("TwinSword");

}

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

sword.GetComponent<MeshCollider>().enabled = true;

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

sword.GetComponent<MeshCollider>().enabled = false;

}

}

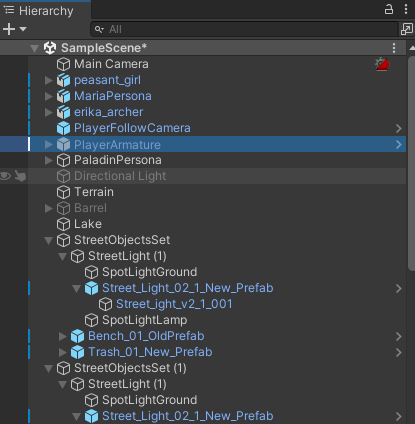
Название "TwinSword" взяли с названия меча в иерархии:



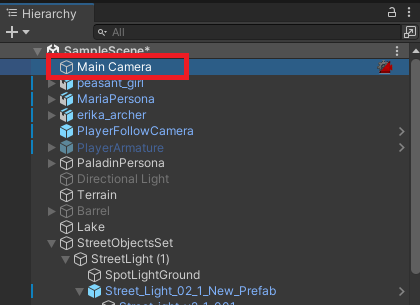
Теперь когда перс будет просто подходить к бочке то она падать не будет, но коогда перс ударит бочку мечем то на мече включится мэш коллайдер и бочка отскочит от него

# Делаем ночь

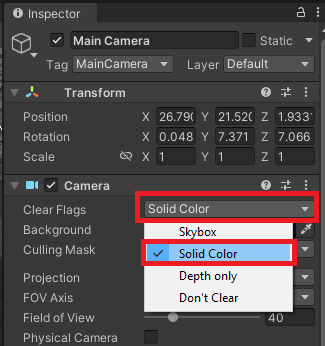
1) отключаем Directional Light



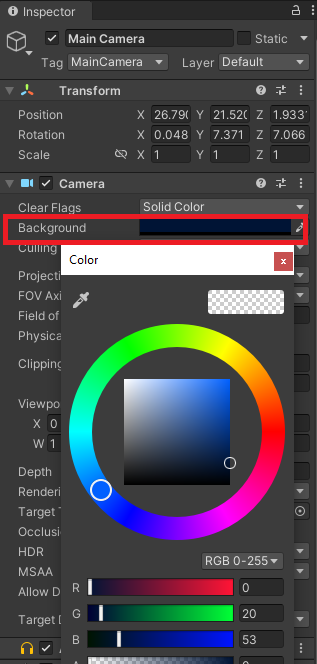
Выбираем главную камеру:



Теперь в инспекторе:

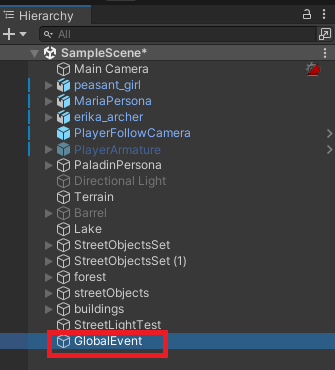
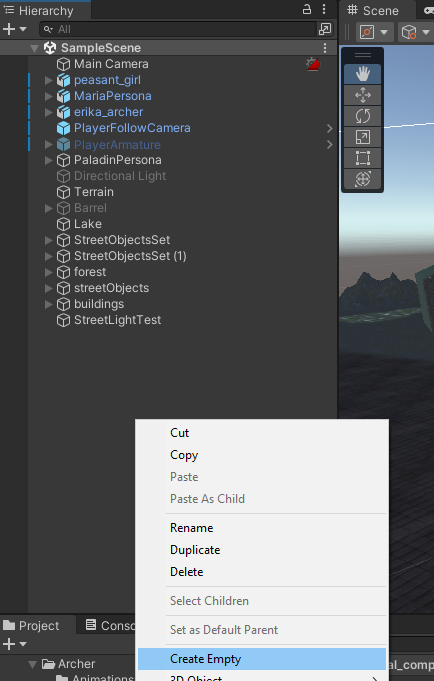


Выбираем цвет:

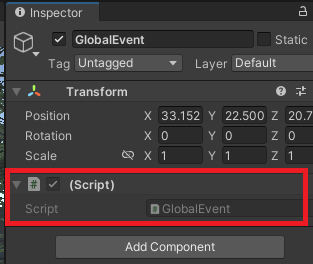


# Переделываем включение/выключение фонаря через событие

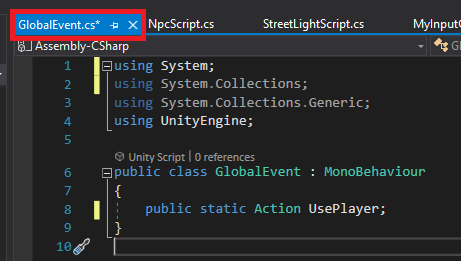
Создаем в иерарчии пустой объект и называем его GlobalEvent:



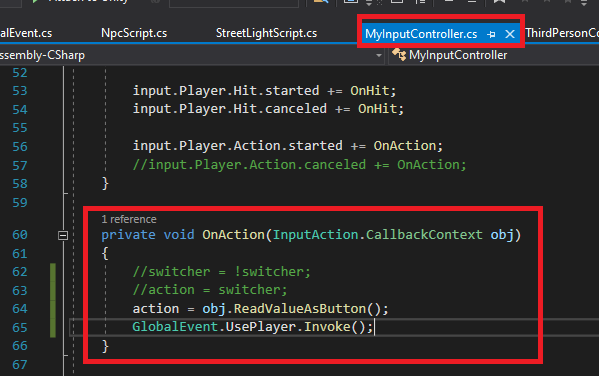
Добавляем ему скрипт с таким же названием:



Пишем в нем:



Теперь меняем логику в инпут контроллере:



И меняем логику в скрипте фонаря:

