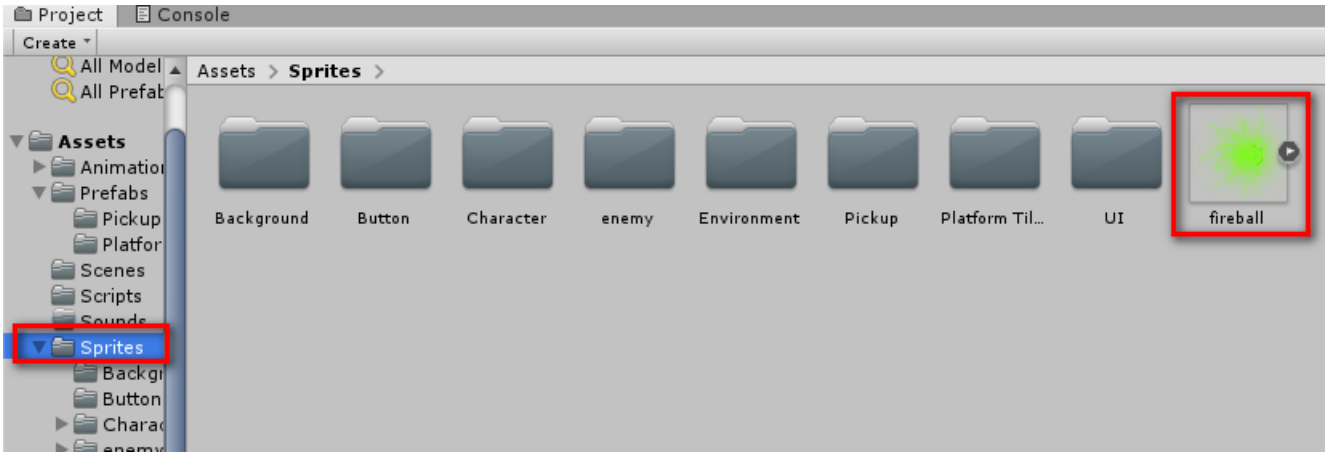
Для початку додамо до проекту спрайт снаряду нашого робота:

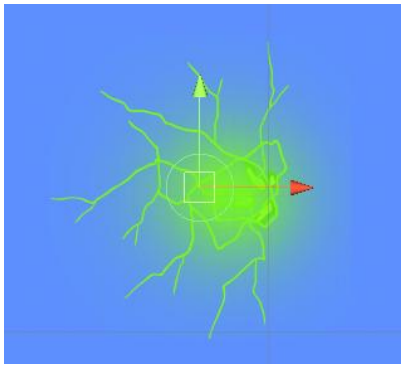


- Встановлюємо для доданого спрайту PixelsPerUnit: 800

- Max size: 512

- Mesh Type: Full rect

Додаємо снаряду круглий колайдер і робимо його тригером



Додаємо снаряду rigidbody2D, заморожуємо обертання осі Z, міняємо Collision Detection на continious і відправляємо в префаби

Створюємо скрипт Fireball і застосовуємо його до снаряду. Створюємо у скрипті три поля:

```
#region fields
[SerializeField] private Rigidbody2D rb2d = null;      //Для задания импульса снаряду
[SerializeField] private float fireballForce = 5f;           //Скорость полета снаряда
[SerializeField] private float lifeTime = 3f;    //Время существования снаряда на экране
#endregion
```

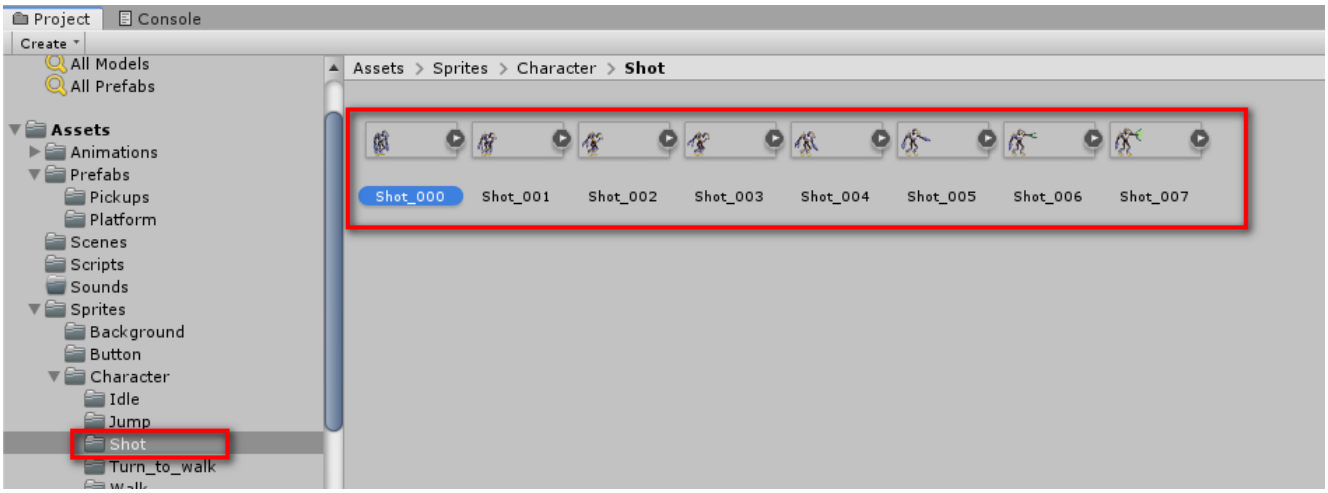
Не забудьте заповнити поле rb2d в інспекторі!

Якщо ми просто створимо снаряд при натисканні на ЛКМ, це буде погано поєднуватися з майбутньою анімацією пострілу. Іншими словами створення снаряда та анімація пострілу

запускатимуться одночасно. Мабуть, снаряд вже вразить ціль, коли анімація пострілу нарешті закінчиться.

Таким чином, постріл буде складатися з двох етапів. На першому етапі запускатиметься анімація пострілу. Після повного її закінчення запускатиметься метод, що створює і запускає снаряд у політ!

Почнемо з першого етапу. Імпортуйте в проект анімацію пострілу:

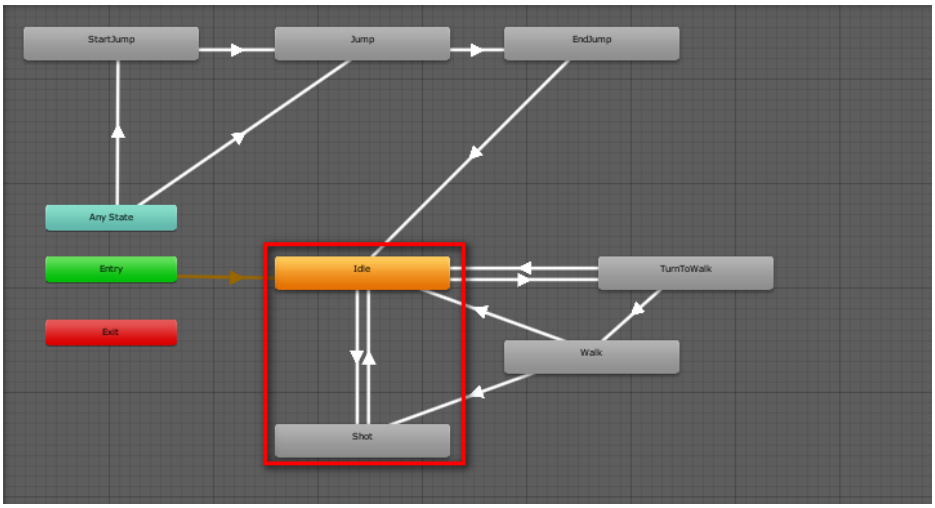


Самостійно підберіть відповідні параметри спрайтів (pixel per unit, max size, mesh type). На жаль, ця анімація не зовсім адаптована для гри, т.к. художник значно розтяг спрайти по осі X. Це означає, що надалі вам доведеться витратити деякий час для підбору pivot.

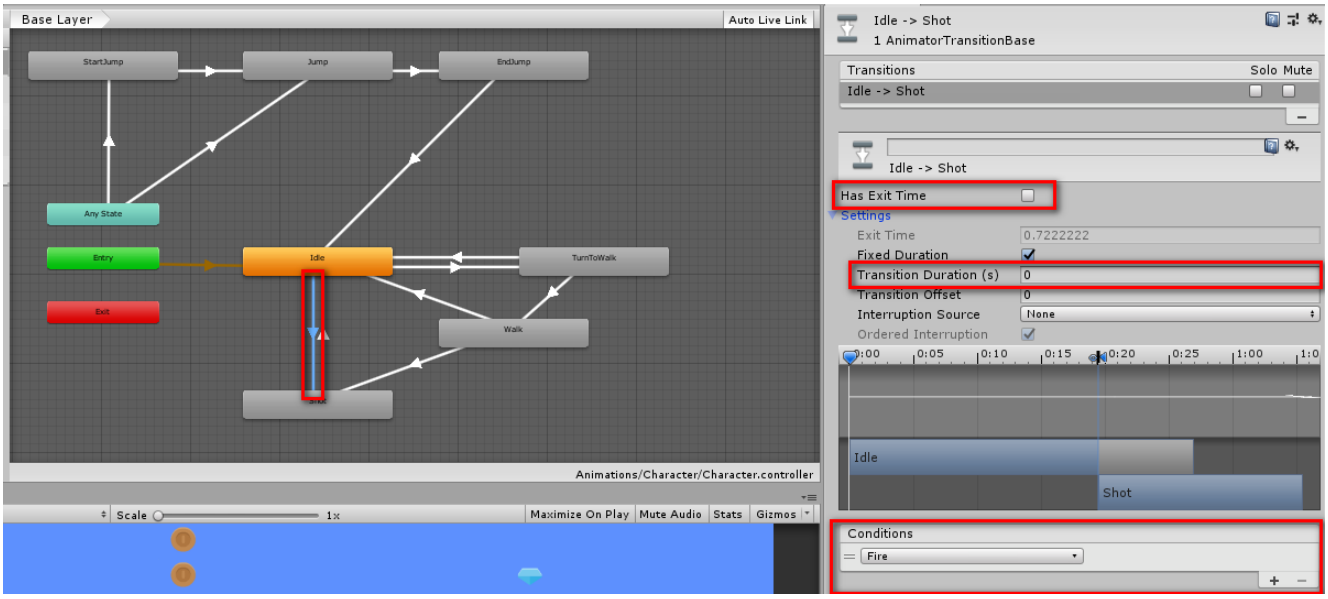
Створіть у робота анімацію Shot і перетягніть у неї імпортовані спрайти:

Значення семплів вам надалі доведеться підібрати самостійно.

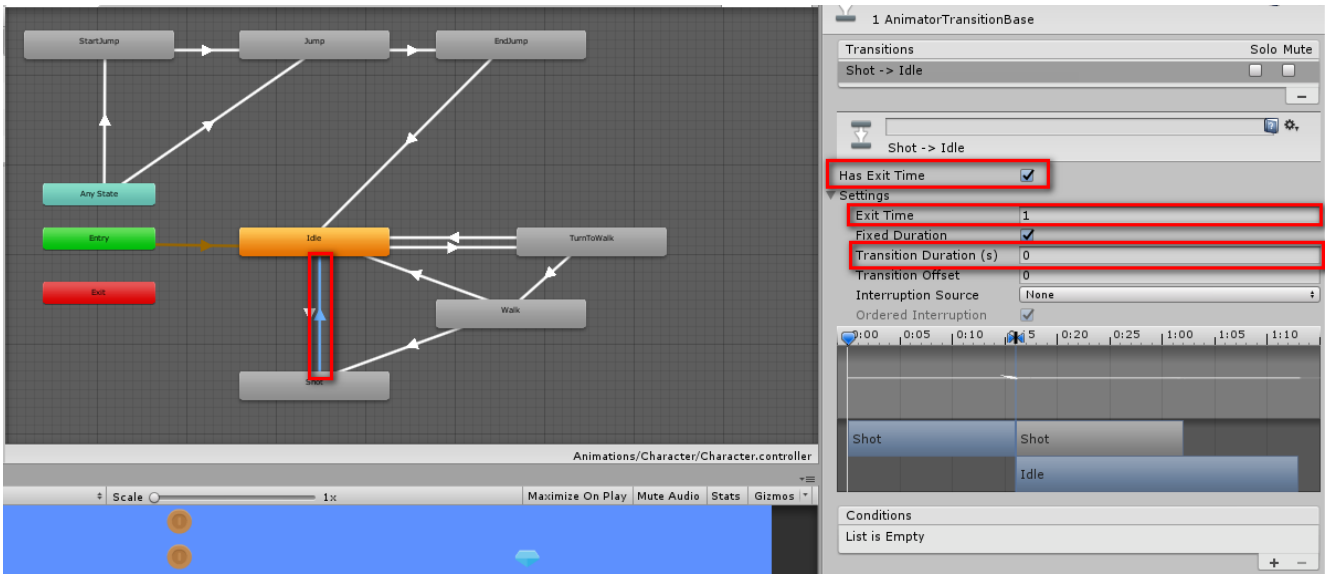
Реалізуємо можливість запуску анімації стрільби зі стану очікування та руху. Для початку зв'яжемо стан очікування та стрілянини:



З очікування на стрілянину переходитимо на виклик тригера (назвемо його «Fire»)



Зі стрілянини в очікування переходитимо після закінчення анімації:



Зауважте, що значення Exit Time ви підбираєте самостійно. Виходячи з того, скільки, на вашу думку, має тривати анімація стрілянини. Наприклад прийнята тривалість анімації 1 секунда. Цей параметр важливий, т.к. у нашій варіації реалізації цей параметр впливає на скорострільність робота!

Реалізуємо запуск анімації. Відкриваємо скрипт Player і створюємо в ньому три додаткові поля:

```
[SerializeField] private GameObject fireball = null;      //Для доступа к снаряду
[SerializeField] private Transform fireballSpawnPoint = null; //Для определения позиции при создании снаряда
[SerializeField] private float shootForce = 5f; //Скорость снаряда
```

Напишемо метод перевіряючий натискання ЛКМ для пострілу і анімацію, що запускає після цього:

```
//Метод создания снаряда при щелчке ЛКМ
1 reference
void Fire()
{
    //0 - ЛКМ, 1 - ПКМ
    if (Input.GetMouseButtonDown(0))
    {
        animator.SetTrigger("Fire");
    }
}
```

Звичайно метод повинен запускатися на кожному кадрі гри. Додаємо його виклик у метод Update:

```
Unity Message | 0 references
private void Update()
{
    Fire();
}
```

Анімація вже має працювати! На цьому етапі, швидше за все, спрацьовувати вона буде не дуже красиво. Або робот при одному клацанні програватиме більше 1 анімації пострілу. Або (що найімовірніше) не буде встигати програвати її остаточно за відведений час. Оперуючи значеннями Exit time і Samples вам потрібно досягти нормального відображення пострілу.

Тепер потрібно реалізувати другий етап, тобто створення та запуск самого снаряду. Після попередніх етапів ви мали зрозуміти, що розрахувати цей момент вкрай складно. Додає складності ще й те, що будь-якої миті ми можемо прискорити або сповільнити анімацію, що призведе до необхідності повторних розрахунків. Однак розробники Unity передбачили таку ситуацію. Ми можемо не тільки запускати анімацію по сигналу з методу, а й запускати метод за

сигналом з анімації. Іншими словами, після програвання останнього кадру анімації пострілу ми подамо сигнал про необхідність створити і запустити снаряд.

Для початку реалізуємо метод, що запускає снаряд у політ. Для цього відкриваємо скрипт Fireball і пишемо:

```
/// <summary>
/// Заданием импульса снаряду
/// </summary>
/// <param name="forceDirection">Направление и сила натяжения тетивы</param>
/// <param name="parent">Объект, который производит выстрел</param>
1 reference
public void SetImpulse(float forceDirection, GameObject parent)
{
    //Если значение силы передано отрицательным, стрела должна лететь влево
    if (forceDirection < 0)
    {
        //Разворачиваем объект по оси Y
        transform.rotation = Quaternion.Euler(0, 180, 0);
    }


    rb2d.AddForce(Vector2.right * fireballForce*forceDirection, ForceMode2D.Impulse);

}
```

Тепер у потрібний момент часу гравець повинен створити снаряд і за допомогою написаного методу запустити його. Для того, щоб вказати конкретну точку створення снаряду, створимо всередині гравця порожній об'єкт з назвою «FireballSpawnPoint»:

Не треба пересувати цю точку з центру персонажа, інакше вам доведеться самостійно шукати спосіб її відображення у разі повороту.

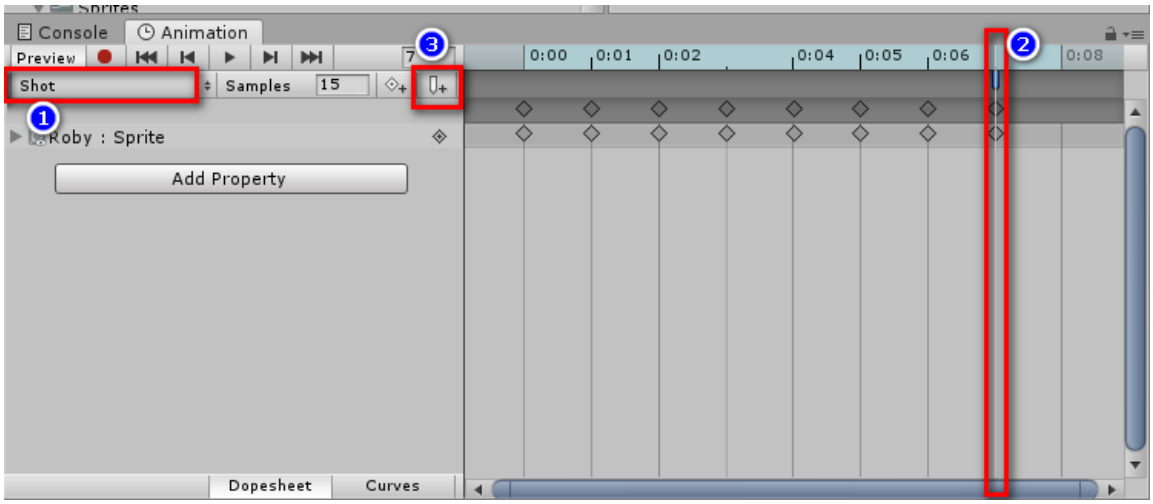
Тепер заповніть в інспекторі FireballSpawnPoint гравця.

У скрипті Player пишемо метод:

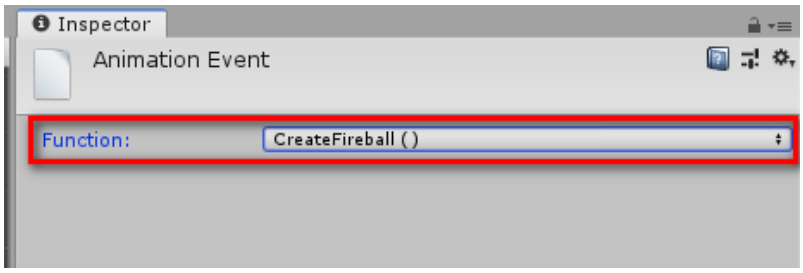
```
0 references
void CreateFireball()
{
    //Создание снаряда в указанной позиции
    //Quaternion.identity - без вращения
    GameObject bullet = Instantiate(fireball, fireballSpawnPoint.position, Quaternion.identity);

    //Выстрел
    bullet.GetComponent<Fireball>().SetImpulse(
        spriteRenderer.flipX ? -shootForce : shootForce,
        gameObject);
}
```

Залишилося визначити момент, коли даний метод повинен запускатися. Як ми вже говорили, це станеться після закінчення анімації Shot. Відкриваємо її, встановлюємо курсор поточного кадру на останній кадр та натискаємо кнопку «Add Event»:



В інспекторі вказуємо функцію, яка має запуститися у цей момент!



Запускаємо та перевіряємо. Снаряд повинен з'являтись після закінчення анімації пострілу.

Оскільки нанесення урону снарядами поки що не реалізовано, бачимо, що вони накопичуються в інспекторі після кожного пострілу. Реалізуємо час існування кожного снаряда.

Відкриваємо скрипт «Fireball»

```
public void SetImpulse(float forceDirection, GameObject parent)
{
    //Если значение силы передано отрицательным, стрела должна лететь влево
    if (forceDirection < 0)
    {
        //Разворачиваем объект по оси Y
        transform.rotation = Quaternion.Euler(0, 180, 0);
    }


    rb2d.AddForce(Vector2.right * fireballForce*forceDirection, ForceMode2D.Impulse);
    StartCoroutine(StartLife());
}

1 reference
private IEnumerator StartLife()
{
    yield return new WaitForSeconds(lifeTime);  //Пауза на время существования снаряда
    Destroy(gameObject);

    yield break;    //Для безопасности
}
```

Тепер снаряди зникають після проходження вказаного часу.

Залишилося реалізувати завдання снарядами шкоди.

Створюємо скрипт «TriggerDamage» та застосовуємо його до снаряду.

Створюємо у скрипті два поля:

```
#region Field
[SerializeField] private int damage;        //Урон от снаряда
private GameObject parent;  //Для сохранение объекта, который стреляет,
                            //чтобы не наносить ему урон
#endregion
```

Не забудьте заповнити значення розміру урону в інспеторі

Додаємо властивість для parent:

```
#region Properties
1 reference
public GameObject Parent
{
    get { return parent; }
    set { parent = value; }
}
#endregion
```

Пишемо метод нанесення шкоди:

```
#region Methods
Unity Message | 0 references
private void OnTriggerStay2D(Collider2D col)
{
    //Блокируем работу скрипта в случае прикосновения снаряда к игроку
    if (col.gameObject == parent)
    {
        return;
    }
    //var - анонимный тип данных
    var health = col.gameObject.GetComponent<Health>();  //Считываем скрипт health у объекта

    Destroy(gameObject);
    if (health != null)
    {
        health.SubstractHealth(damage);
    }
}
#endregion
```

Тепер доповнимо скрипт «Fireball». Додаємо до нього поле:

```
#region fields
[SerializeField] private Rigidbody2D rb2d = null;      //Для задания импульса снаряду
[SerializeField] private float fireballForce = 5f;           //Скорость полета снаряда
[SerializeField] private float lifeTime = 3f;    //Время существования снаряда на экране

[SerializeField] private TriggerDamage triggerDamage = null;   //Для связи со снарядом и защиты от атаки самого себя
#endregion
```

В інспекторі заповнюємо поле.

Доповнюємо метод SetImpulse

```
public void SetImpulse(float forceDirection, GameObject parent)
{
    //Если значение силы передано отрицательным, стрела должна лететь влево
    if (forceDirection < 0)
    {
        //Разворачиваем объект по оси Y
        transform.rotation = Quaternion.Euler(0, 180, 0);
    }

    triggerDamage.Parent = parent;  //Для того, чтобы снаряд не наносил урон игроку
    rb2d.AddForce(Vector2.right * fireballForce*forceDirection, ForceMode2D.Impulse);
    StartCoroutine(StartLife());
}
```

Тепер снаряд завдає шкоди ворогам!