



GameDesign. Unity.



<Developer/>

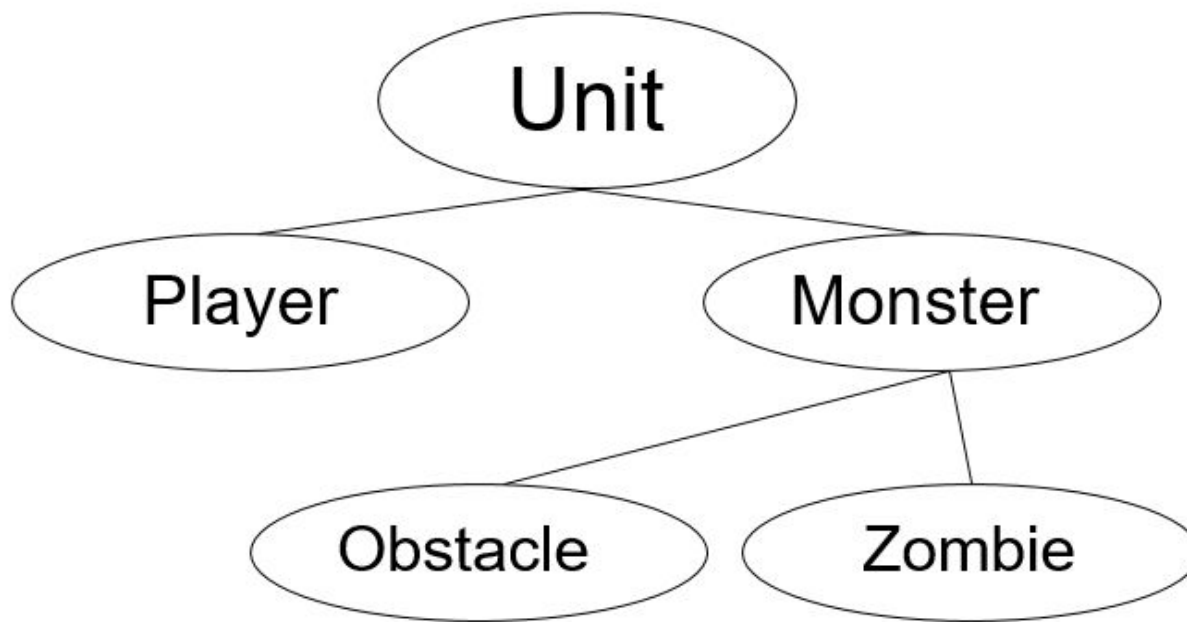
**Меню, анімація, левелінг. Запис
рекорду в файл.**



Повторення

Що таке клас в ООП?

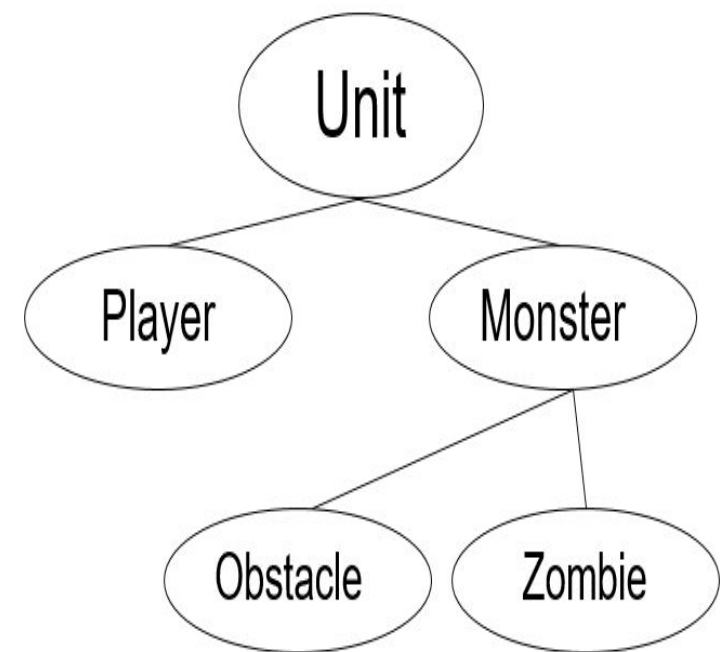
Класи - це, по суті, шаблони, за якими можна створювати об'єкти. Кожен об'єкт містить дані та методи, котрі маніпулюють цими даними.



Повторення

Що таке наслідування?

Наслідування – процес, завдяки якому один клас може отримати (наслідувати) властивості іншого класу і додавати риси характерні тільки для нього самого.



```
public class Monster : Unit{}
```

```
public class Zombie : Monster {}
```

Повторення

Як реалізувати обмін даними для системи Scoring?

Слід створити скрипт **ScoreManager**, через який відбудеться обмін даними і створити в ньому *public* змінну:

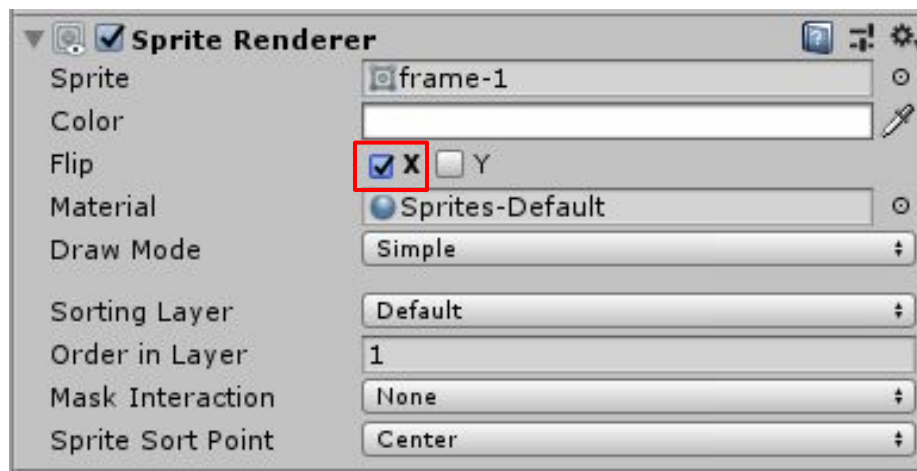
```
public static int score;
```

Для того аби отримати змінну за межами класу, змінна повинна бути загальнодоступною і звертатися до неї слід через ім'я класу:

```
ScoreManager.score += 100;
```

Анімація

На даному етапі, не залежно від напрямку руху, наш персонаж дивиться лише вправо. Щоб при русі вліво він дивився в ту ж сторону, слід його відобразити по осі x (flip x):



Анімація

В скрипті `_player` створимо об'єкт класу `SpriteRenderer`:

```
private SpriteRenderer sprite;
```

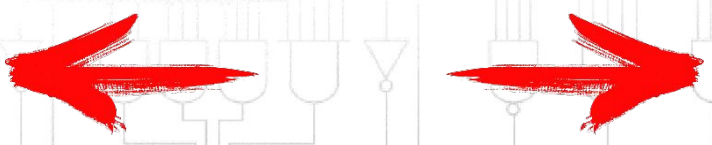
В методі `void Awake()` передамо створеному об'єкту компоненти `SpriteRenderer` :

```
sprite = GetComponent<SpriteRenderer>();
```

Оскільки в методі `Run()` відбувається зміна напрямку переміщення персонажу, то в нього слід внести незначні зміни:

```
sprite.flipX = direction.x < 0.0f;
```

$direction.x < 0.0f$



$direction.x > 0.0f$

Анімація

Слід змінити ще скрипт **_bullet**, щоб, залежно від повороту персонажу, снаряди летіли вправо/вліво:

```
private Vector3 direction;
```

Тепер ми не задаємо сталий напрям, а будемо змінювати його за допомогою властивості *Direction*. Для цього в скрипті **_player** змінимо метод *Shoot()*:

```
private void Shoot() {  
    _bullet temp = Instantiate(bullet, new  
    Vector3(transform.position.x+ (sprite.flipX ? -1.5f : 1.5f),  
    transform.position.y, 0), bullet.transform.rotation) as _bullet;  
  
    temp.Direction = temp.transform.right*(sprite.flipX ? -1 : 1);  
}
```

Тернарна умовна операція <Developer/>

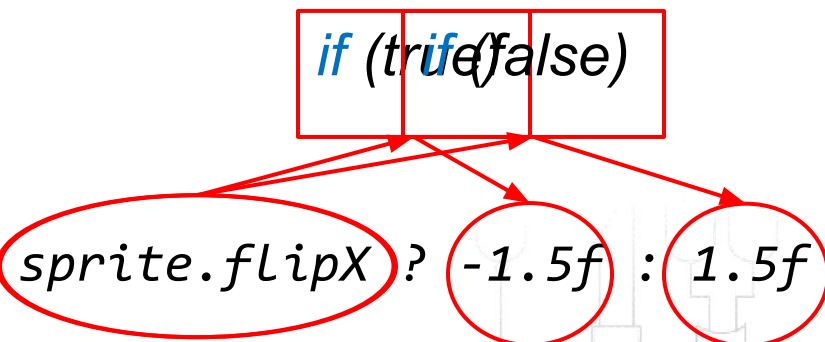
Тернарна умовна операція — в багатьох мовах програмування операція, яка повертає свій другий або третій операнд в залежності від значення логічного виразу, заданого першим операндом :

`condition ? value_if_true : value_if_false`

`a < b ? a : b`

Аналогічно

```
if(a < b)
    return a;
else
    return b;
```



Анімація

Якщо персонаж повернутий вліво (`sprite.flipX = true`) – генеруємо об'єкт `bullet` з відступом `-1.5f` (за персонажем):

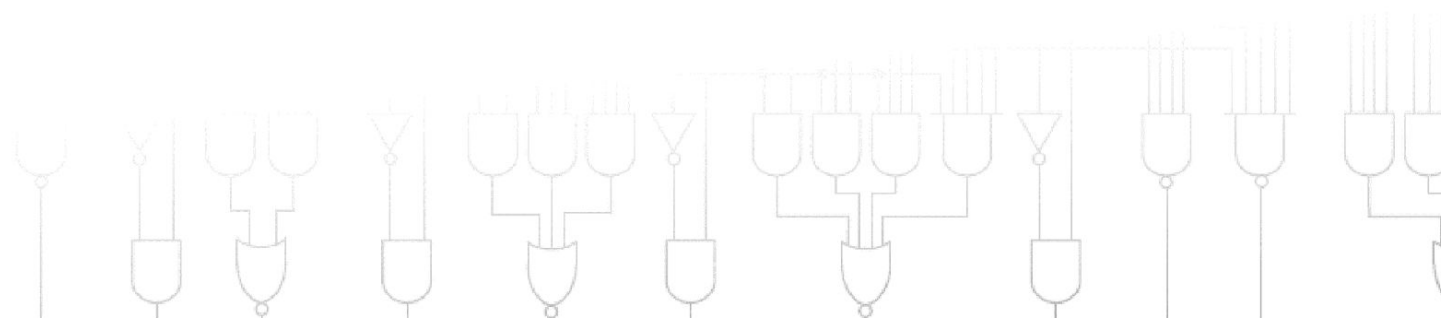
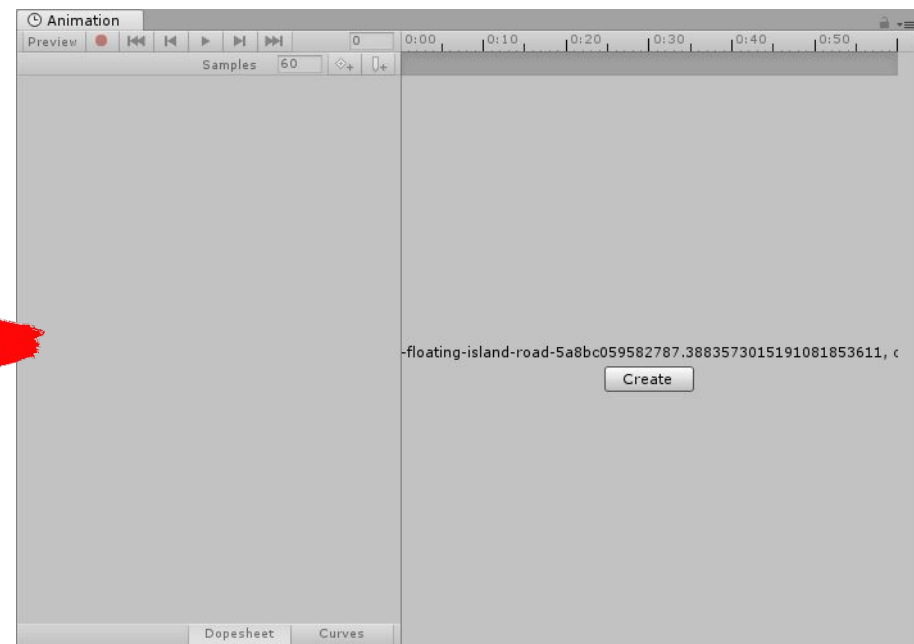
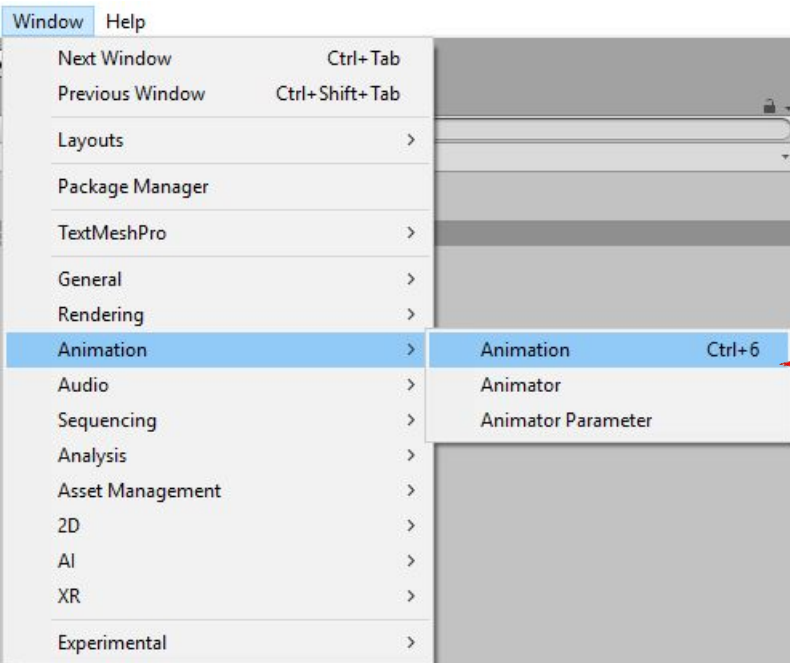
```
_bullet temp = Instantiate(bullet, new  
Vector3(transform.position.x + (sprite.flipX ? -1.5f : 1.5f),  
transform.position.y, 0), bullet.transform.rotation) as _bullet;
```

Якщо персонаж повернутий вліво (`sprite.flipX = true`) – змінимо метод `Direction` на `-1`, звернувшись до нього через об'єкт `temp`:

```
temp.Direction = temp.transform.right * (sprite.flipX ? -1 : 1);
```

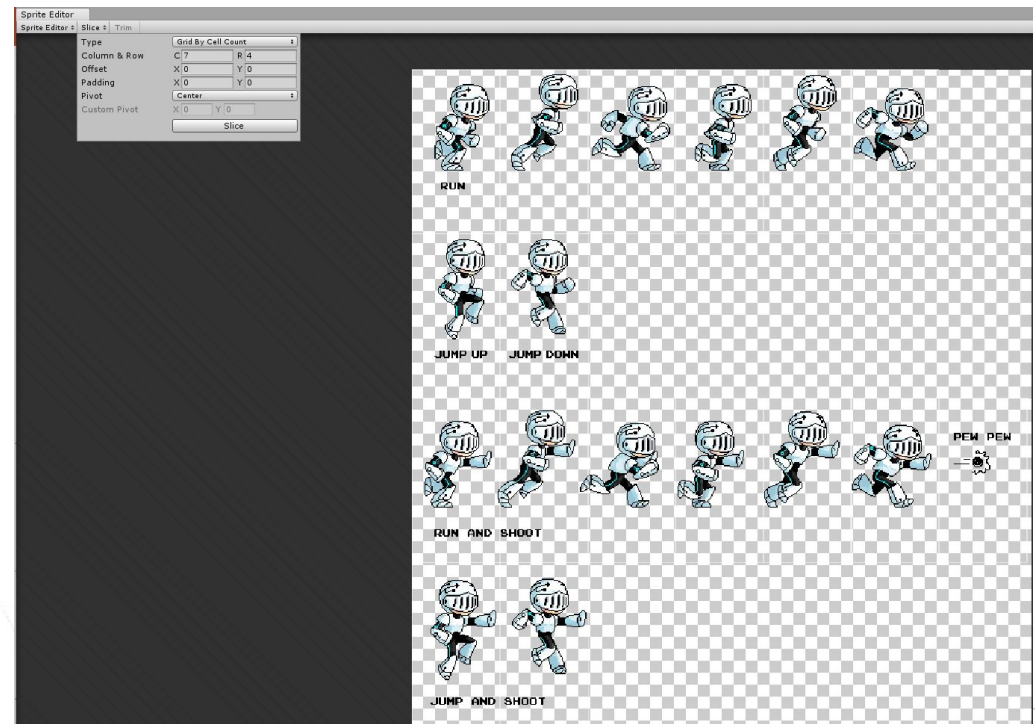
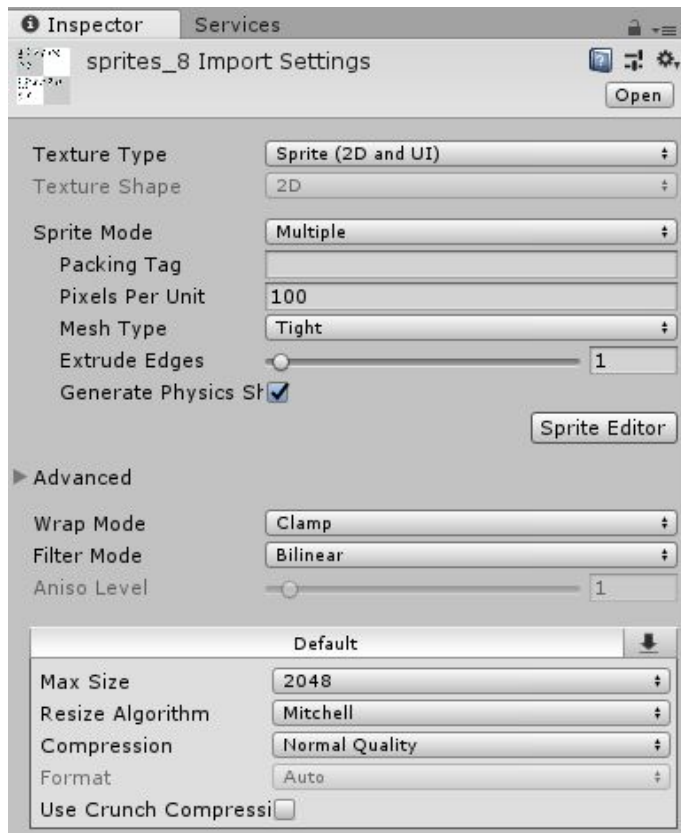
Анімація

Виділимо нашого героя і відкриємо вікно анімації:



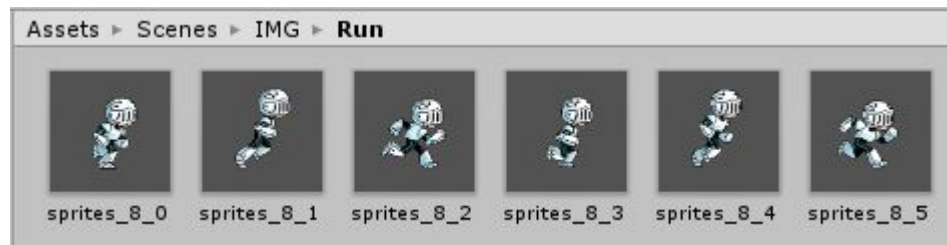
Анімація

Скачати заготовки для анімації героя можна із сайту opengameart.org, або ж скачати SpriteSheet і за допомогою Sprite Editor «порізати» його:

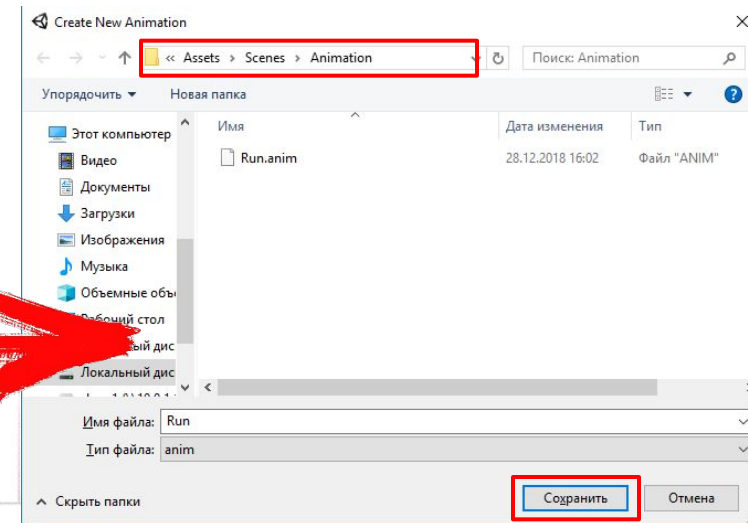
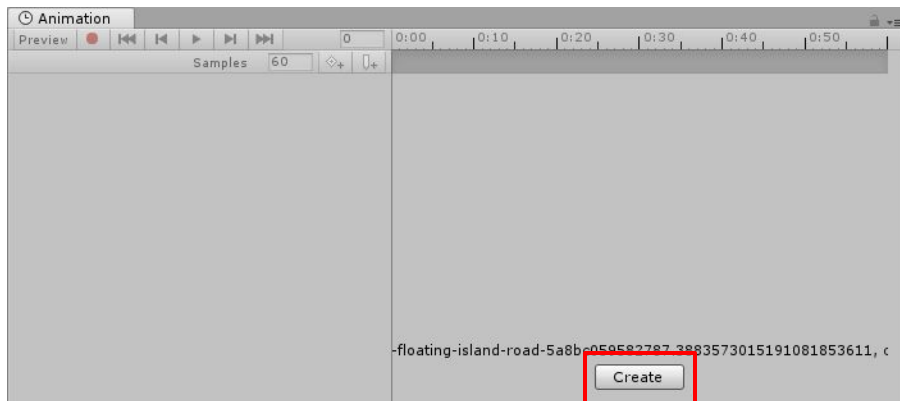


Анімація

Після того, як ви порізали на слайди – розділіть їх по окремим папкам (*Run, Jump, Shoot, тощо*):

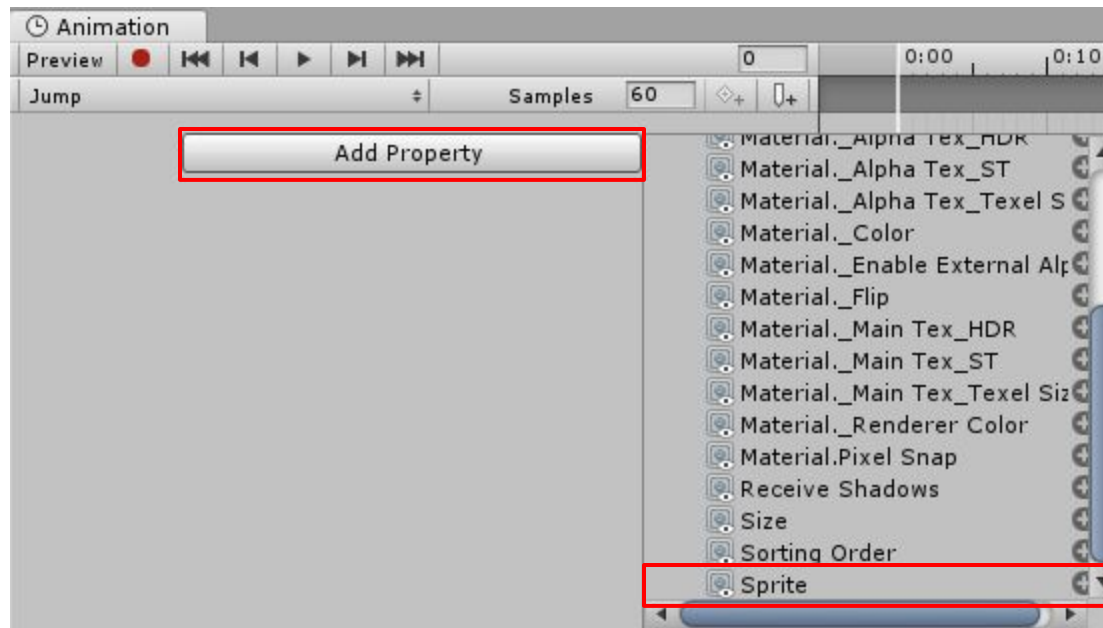


У вікні анімації натисніть *Create* та оберіть місце, для зберігання кліпів анімації:



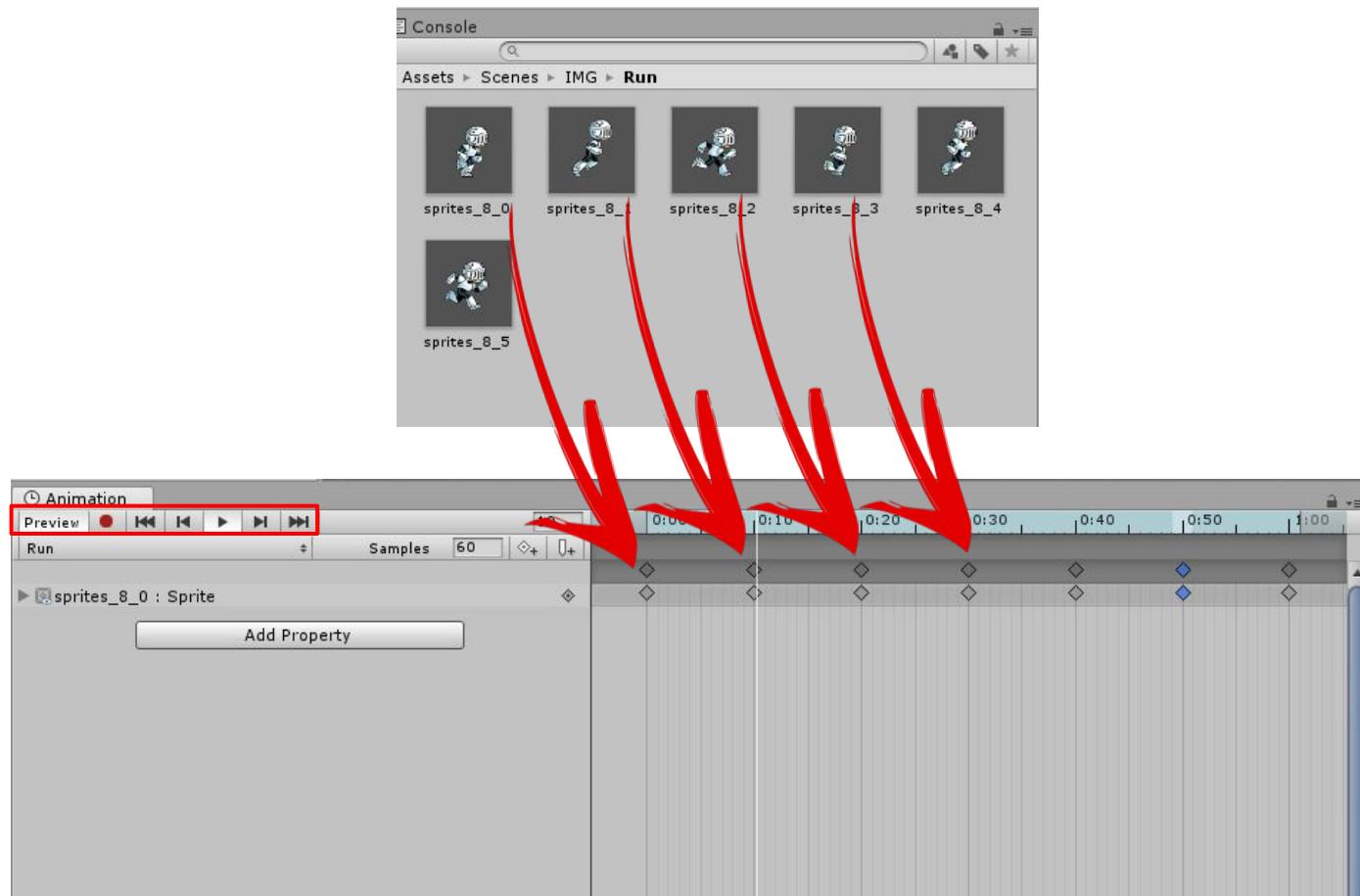
Анімація

Натисніть *Add Property* та, прокрутивши вниз, оберіть *Sprite* :



Анімація

Перетягніть слайди по чергово на часову шкалу. Натисніть play для перегляду:



Анімація

На герої з'явиться компонент Animator:



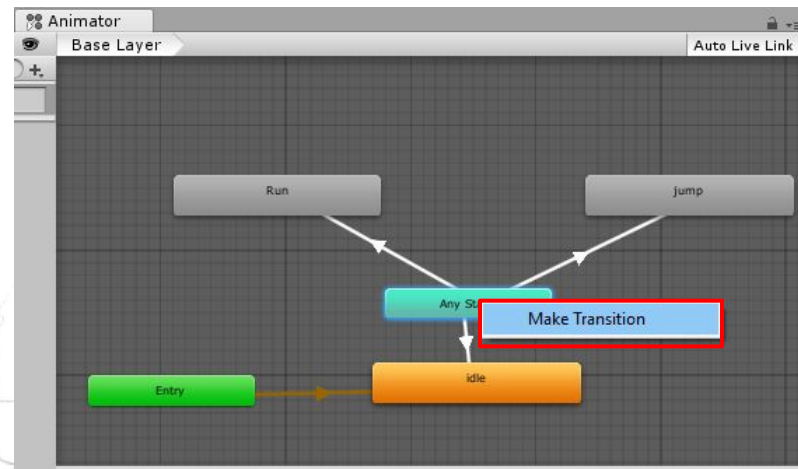
Анімація

У вас в папці Animations має вийти декілька файлів анімації та Animator:



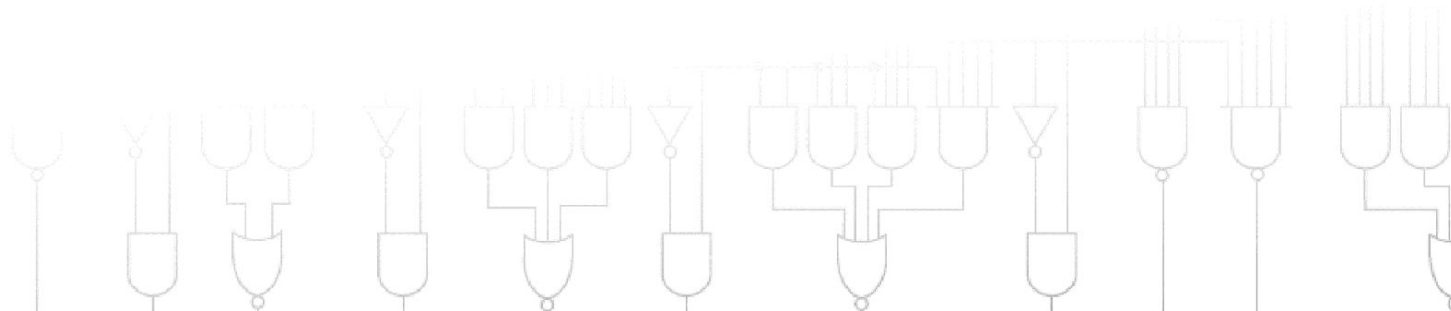
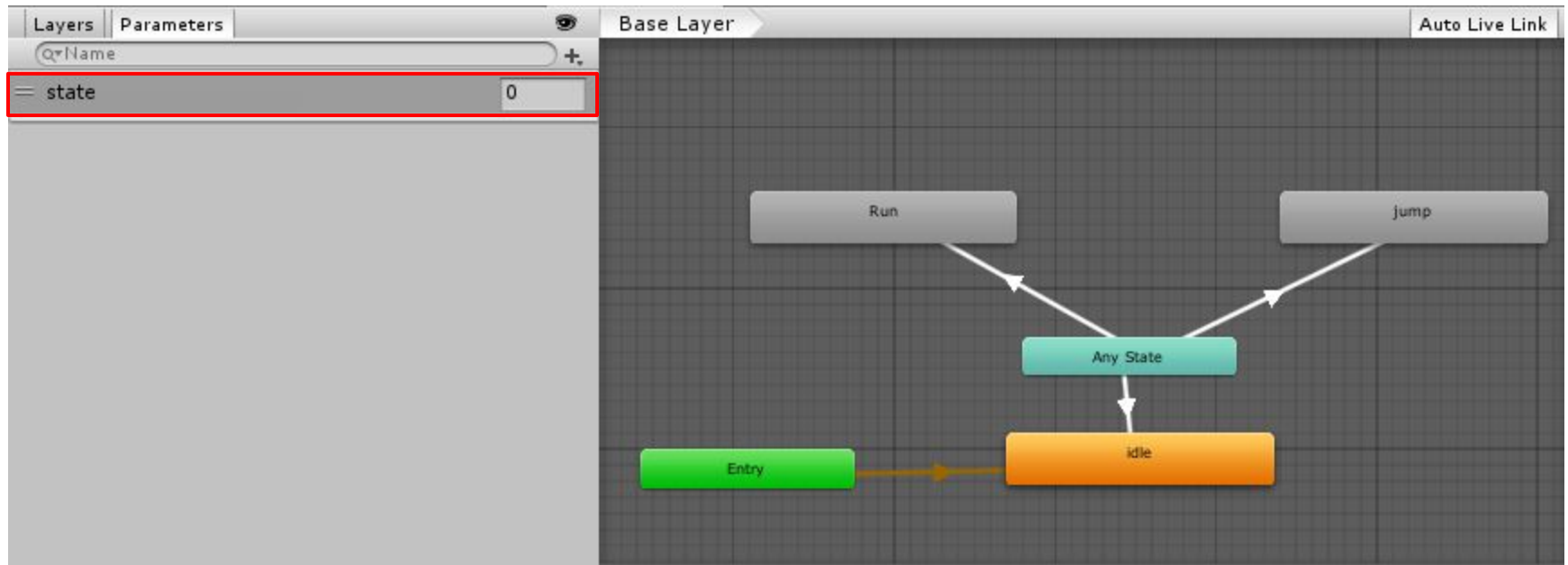
Подвійним кліком відкрийте аніматор, ви побачите створені вами анімації, задайте переходи між ними через блок Any State (ПКМ ->

Make Transition):



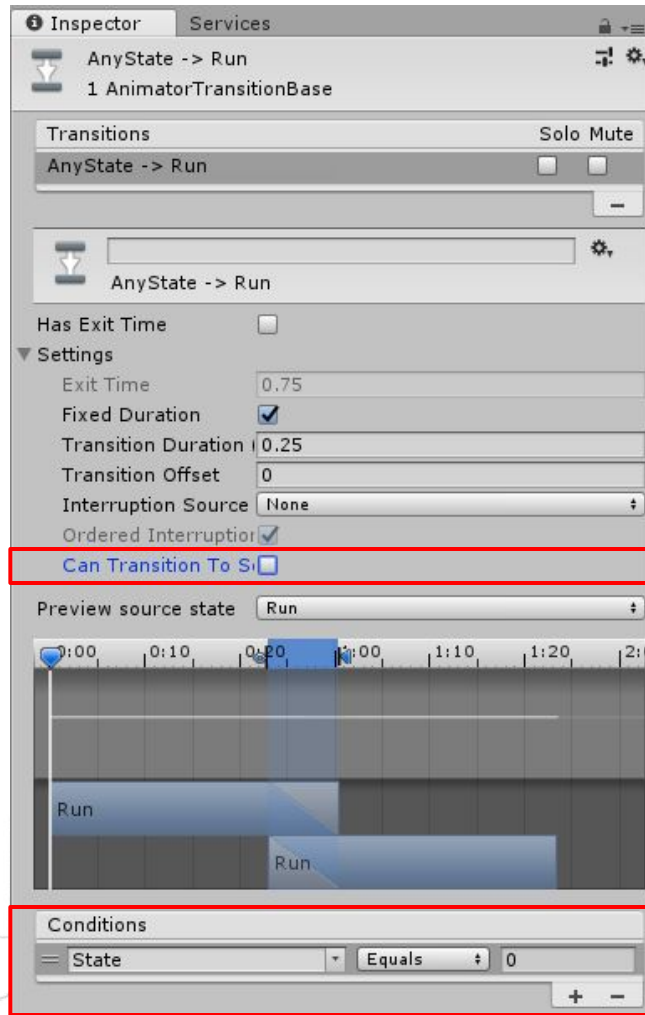
Анімація

Нам слід створити керуючий параметр state:



Анімація

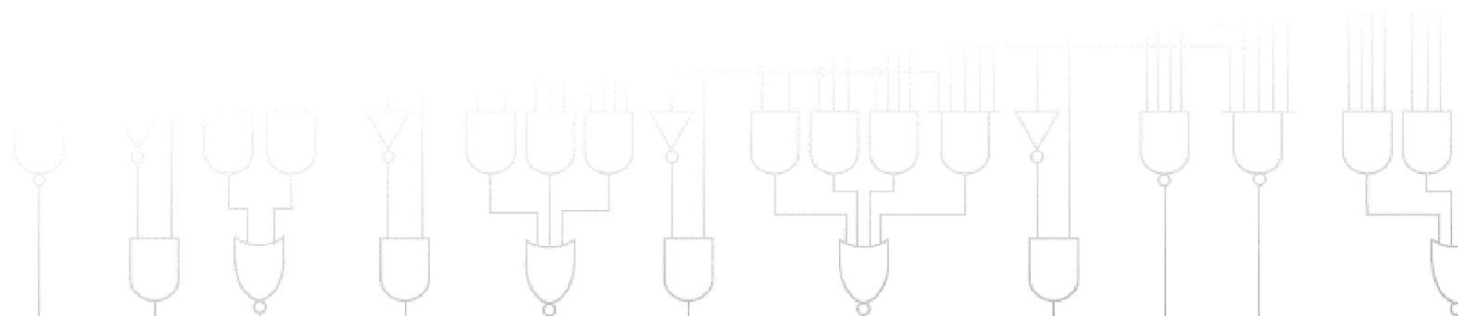
Виділіть стрілочку, та задайте умову переходу в стан Idle - State = 0 і зніміть галочку із Can Transition to Self:



Анімація

В скрипті `_player` на початку створюємо об'єкт `Animator`, в методі `Awake` передаємо йому компоненти аніматора:

```
private Animator animator;  
  
void Awake()  
{  
    animator = GetComponent<Animator>();  
}
```



Анімація

В скрипті `_player` (поза класом!!), додамо перелік (*enumerator*):

```
public enum CharState
{
    Idle,
    Run,
    Jump
}
```

Enumerator присвоїть константі `Idle` – значення `0`, `Run` – `1`, `Jump` – `2`. Створіть уже всередині класу, на початку програми, властивість `State`:

```
private CharState State
{
    get { return (CharState)animator.GetInteger("state"); }
    set { animator.SetInteger("state", (int)value); }
}
```

Анімація

Змінимо метод `FixedUpdate`:

```
if (isTouchingGround) State = CharState.Idle;  
else State = CharState.Jump;
```

Тобто, якщо торкаємось землі – стан `Idle`, в іншому випадку – стан `Jump`.

Аналогічно при виклику функції `Run()` змінимо стан на `Run`:

```
if (Input.GetButton("Horizontal"))  
{  
    Run();  
    if (isTouchingGround)  
    {  
        State = CharState.Run;  
    }  
}
```

Завдання



На прикладі попередніх уроків зробіть скоринг та перехід між рівнями.

