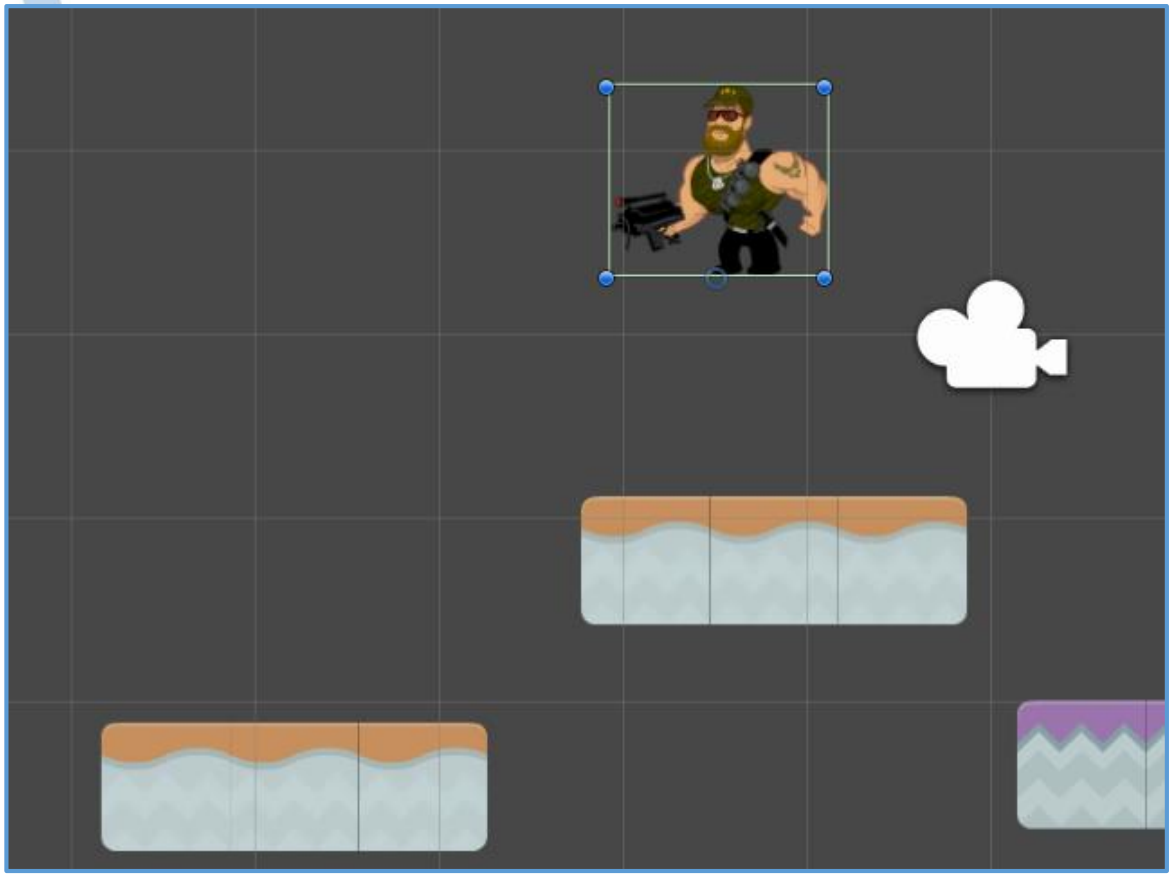
2D Character Scripting

Movers and Shakers

CONTENTS



SCRIPTING FOR 2D

Most of what you know and love about scripting of 3D objects in Unity is the same. The primary difference that you will discover is the name of components has '2D' added to the end. There are, of course, more differences than that, but we will address them as we encounter them.

Scripting a 2D Character

It is now time to write a script to move our character around the level. The two main features our character should have is movement and jumping. We will implement each of these in turn.

Character Movement

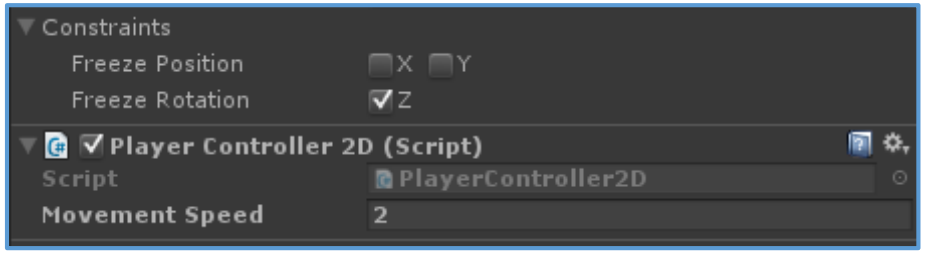
Our first task is to get our character moving left and right on the screen.

1. Create a script called PlayerController2D and attach it to the character you placed in your level in the previous handout.
2. Add a public float variable called 'MovementSpeed' to the script to the script.
3. Add a private Rigidbody2D variable called 'rb' to the script.
4. Add a private float variable called 'moveInput'.

These 3 variables should be familiar to you. Input, speed and a Rigidbody are the 3 things that are common when first setting up most characters to move in Unity.

```
public float MovementSpeed;

Rigidbody2D rb;
float moveInput = 0.0f;
```



5. Add an Awake function to the script.
6. Get the Rigidbody2D component from the player and store it in the ‘rb’ variable.

```
void Awake()
{
    rb = GetComponent<Rigidbody2D>();
}
```

This is yet another familiar step, with the exception of 2D being added to the Rigidbody.

Next, we need to check our inputs.

7. Add an Update function to the script.
8. Get input from the “Horizontal” axis and store it in the ‘moveInput’ variable.

```
void Update()
{
    moveInput = Input.GetAxis("Horizontal");
}
```

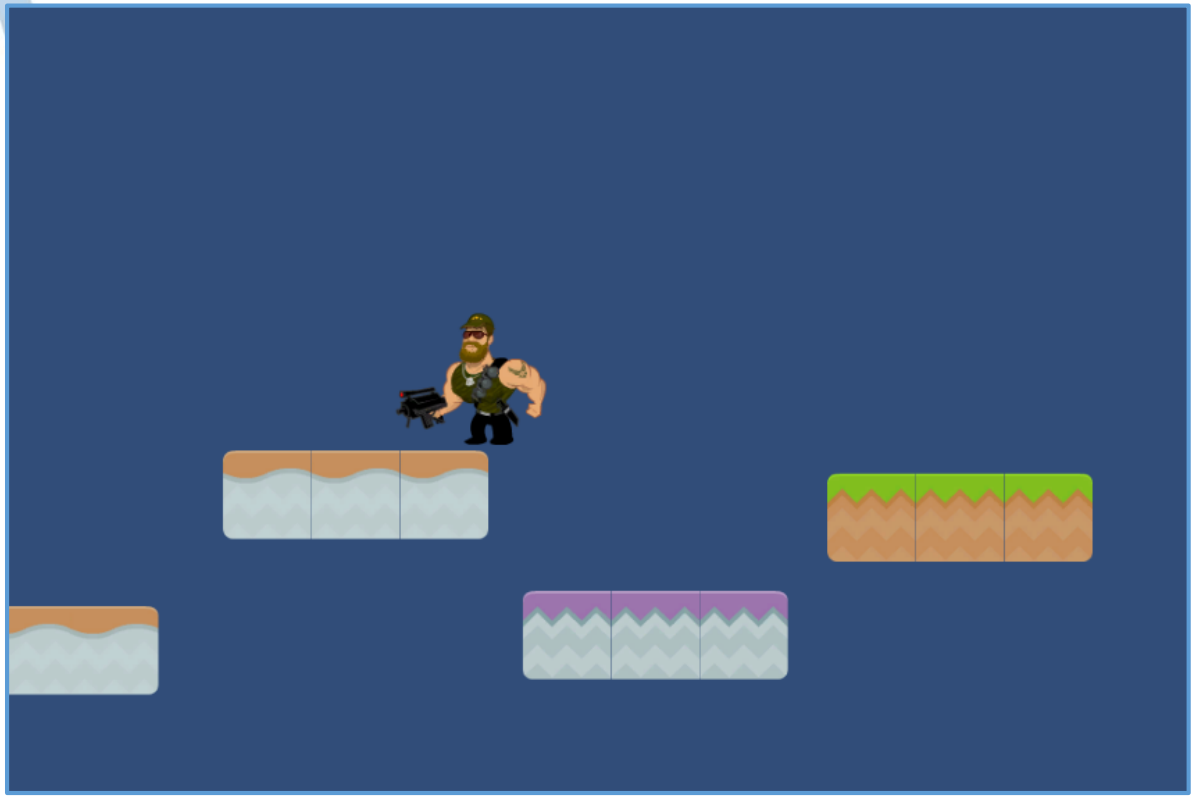
As you will see, we check our inputs inside of Update, but move our character inside of FixedUpdate. Inputs only change each frame. The Update function is run each frame. This is an ideal place to check if we have pressed any buttons or moved any sticks.

9. Add a FixedUpdate function to the script.
10. Get the current velocity of your player from the Rigidbody2D and store it a Vector2 variable called vel.
11. Multiply ‘moveInput’ by ‘MovementSpeed’ to get how fast the character should be moving on-screen, then store the result in the x-component of ‘vel’.
12. Store the newly updated ‘vel’ back into the velocity of the Rigidbody2D.

```
void FixedUpdate()
{
    Vector2 vel = rb.velocity;
    vel.x = moveInput * MovementSpeed;
    rb.velocity = vel;
}
```

This is, of course, not a physically accurate way to move something. For a character we are controlling, however, it's perfectly fine. We want precise control, not physically accurate control.

Jump back into Unity, select your character and set a movement speed. Now you're all set to test out your movement! Don't forget to set a **movement speed** on the character, otherwise your character won't move.



2D Character Jumping

With moving our character complete, we can now try to get jumping working. This is going to require a bit more work.

13. Add some public variables to the top of the script.
    a. Add a float variable called JumpSpeed.
    b. Add a LayerMask variable called GroundMask.
    c. Add a Transform variable called FeetTopLeft.
    d. Add another Transform variable called FeetBottomRight.

```
public float MovementSpeed;
public float JumpSpeed;
public LayerMask GroundMask;
public Transform FeetTopLeft;
public Transform FeetBottomRight;
```

We need quite a few variables to set up correct jumping. We need to provide a speed for our jump, that much is clear. The purpose behind rest of these variables will become clear as we go.

14. Add a private bool variable called jumpInput to the top of the script.
15. Add a private bool variable called justJumped to the top of the script.

```
Rigidbody2D rb;
float moveInput = 0.0f;
bool jumpInput = false;
bool justJumped = false;
```

Inputs are also important!

With these variables in place, we will return to Unity and get them appropriately set up.

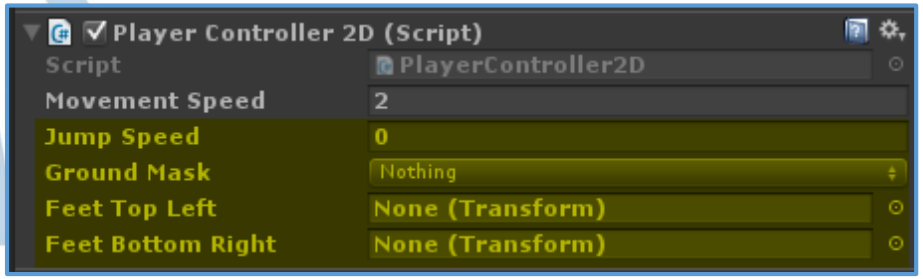
16. Set a speed for your jump. Anything over 10 should work nicely.

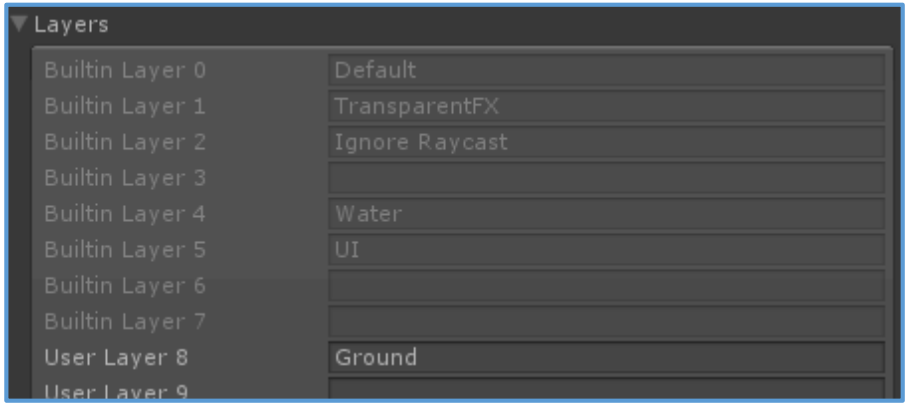
To set up the Ground Mask, we will need to create a new layer for the platforms to be on. We need to know when our character is standing on a platform, so we know that we are standing on something we can jump off of.

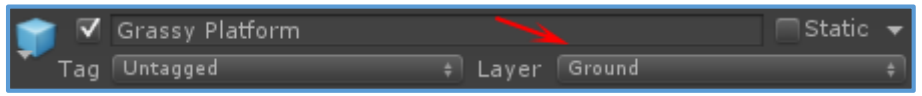
17. Open the Tags and Layers inspector. This can be found in Edit->Project Settings->Tags and Layers.
18. Open the Layers section.
19. In the box labelled 'User Layer 8', type "Ground" without the quote marks and hit Enter.

With that set up, we can now set up the Ground Mask on our character and all of our platforms to be on the new layer we set up.

20. Select your character, open the Ground Mask drop-down and click on the option labelled 'Ground'.
21. For each platform prefab you made, change its layer to 'Ground'.



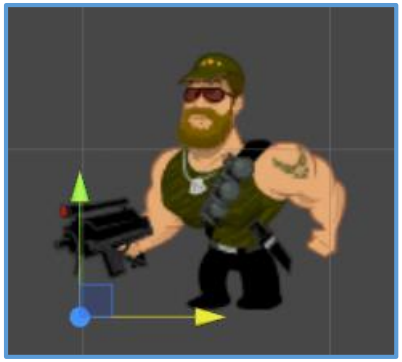


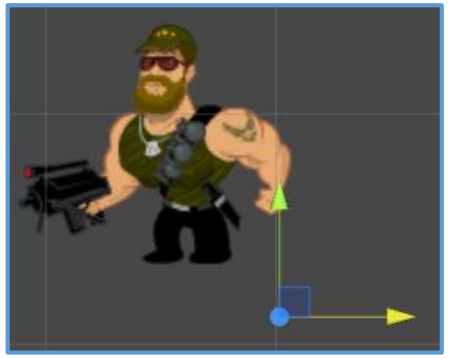


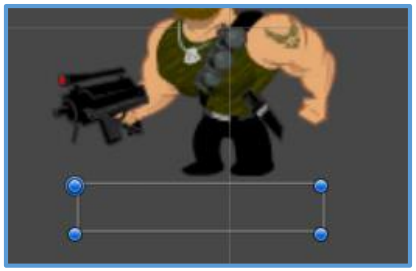
All that is left is too fill in the 'Feet Top Left/Bottom Right' boxes. These two transforms are going to form a rectangle at our character's feet. If this rectangle intersects with one of our platforms, then we will consider our character's feet planted and we can jump.

22. Right-click on your character in the hierarchy and select 'Create Empty'. Do this twice, so you have two objects attached to your character.
23. Name one of them "FeetTopLeft" and the other "FeetBottomRight".
24. Position each of these objects appropriately at your player's feet.

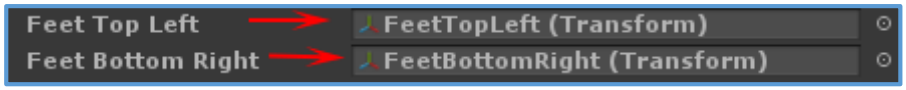
As was mentioned before, these two points will form a rectangle at the player's feet. If you'd like to see that rectangle, select the two "Feet" objects and switch to the Rect Tool (the button in the top-left that is a square with dots in the corners).







25. Select your character and click-and-drag each “Feet” object into its appropriate slot.



We now have everything in place within Unity, let’s return to our script and put all of that set up to use.

26. In the Update function, check if the “Jump” button is pressed down and store that in the jumpInput variable.

```
void Update()
{
    moveInput = Input.GetAxis("Horizontal");
    jumpInput = Input.GetButton("Jump");
}
```

27. In the FixedUpdate function, add an if statement in-between where you set the x-component of vel and where you set it into the Rigidbody rb. In the if statement, check if jumpInput is true.
    a. If so, set the y-component of vel to JumpSpeed.

```
void FixedUpdate()
{
    Vector2 vel = rb.velocity;
    vel.x = moveInput * MovementSpeed;
    if(jumpInput == true)
    {
        vel.y = JumpSpeed;
    }
    rb.velocity = vel;
}
```

Try this out and you will find that you can fly by holding down the spacebar (depending on the strength of your jump). This is obviously not what we want in a typical platformer, so we will use those “Feet” objects to address this.

28. Add 2 extra conditions to the if statement we wrote previously. If jumpInput is true, justJumped is false and the box created by our two “Feet” objects intersects with something on the ‘Ground’ layer, then allow the character to jump.

```
if(jumpInput == true && !justJumped &&
    Physics2D.OverlapArea(FeetTopLeft.position, FeetBottomRight.position, GroundMask) != null)
{
    vel.y = JumpSpeed;
    justJumped = true;
}
```

29. Add another if statement after the first that checks if jumpInput is false, then set justJumped to false.

```
if(jumpInput == true && !justJumped &&
    Physics2D.OverlapArea(FeetTopLeft.position, FeetBottomRight.position, GroundMask) != null)
{
    vel.y = JumpSpeed;
    justJumped = true;
}
if(jumpInput == false)
{
    justJumped = false;
}
```

By checking the area at the character's feet, the moment they are standing on a platform, you'll be able to jump again.

You should now have a functional platformer!

If you find you can't jump sometimes, check the "Feet" objects cover a large enough area beneath your character.