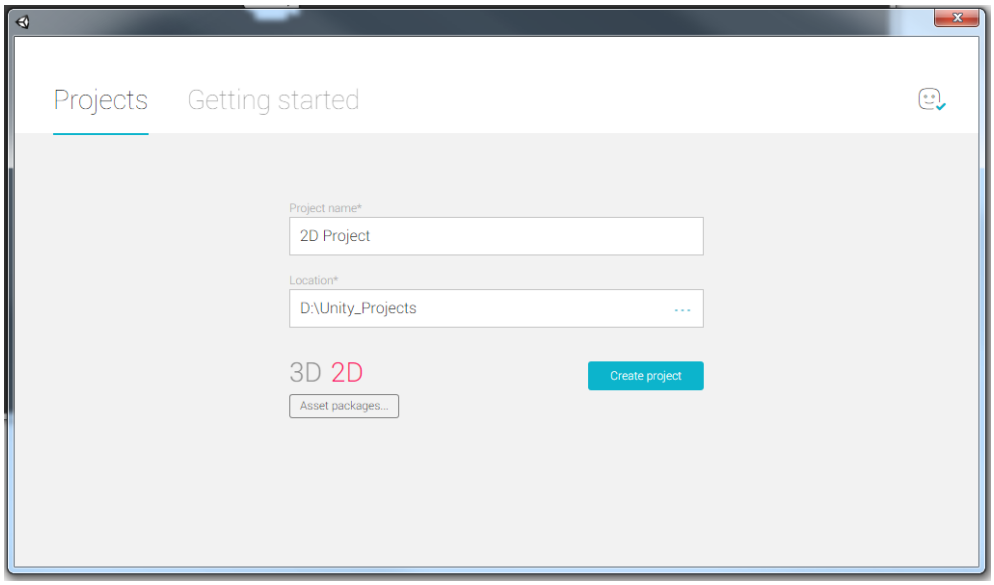
Tutorial – Sprites

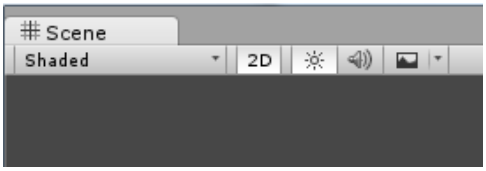
In this tutorial we will be building a simple 2D platformer to get used to using the 2D tools provided by Unity. We'll be making a main character that can jump and shoot, and building a simple level for them to move around in.

Setting up the Project

We'll be making a new project like we have before, however this time we will be selecting 2D in the new project menu. Make sure to add the vs tools to the project as well by clicking assets->import package->Visual Studio Tools

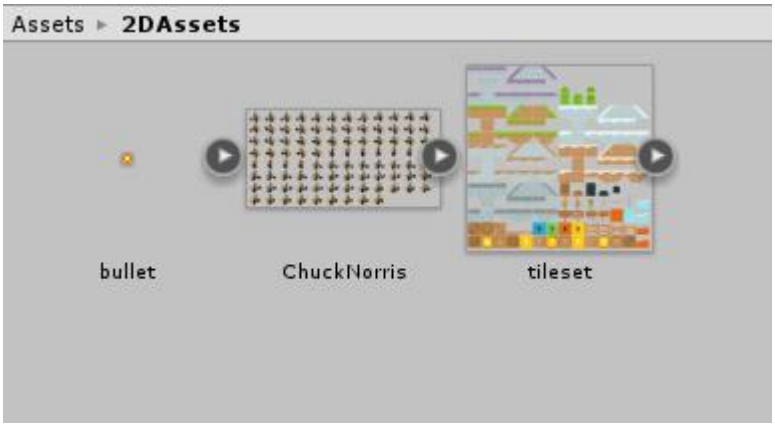


By default your scene view will be setup in 2D mode. This can easily be toggled by the 2D button in the top of the scene window.

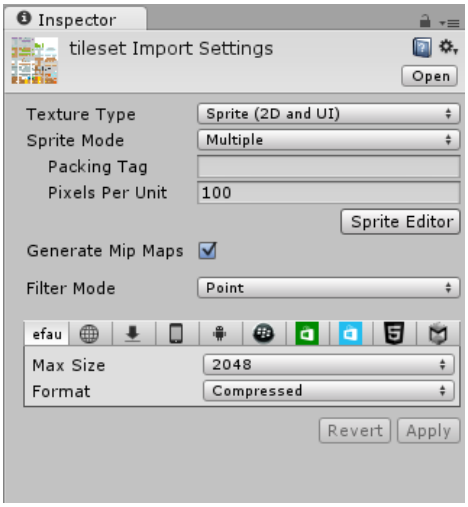


There are a few other differences between the default 2D and default 3D scenes. There's no light in the scene by default. The camera also doesn't have a skybox; it just clears the screen to a constant background colour.

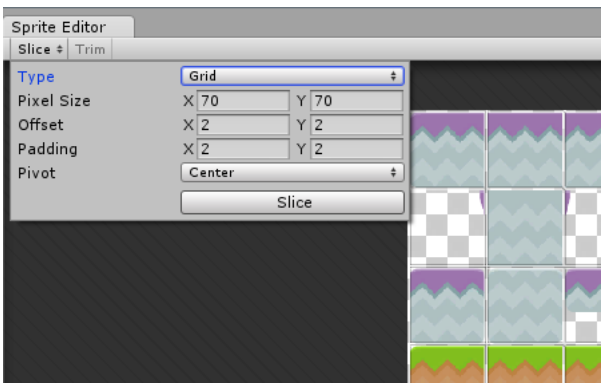
There is a 2D assets unity package file on the portal page for you to use for this containing sprites for a main character, a bullet, and a tileset for the world.



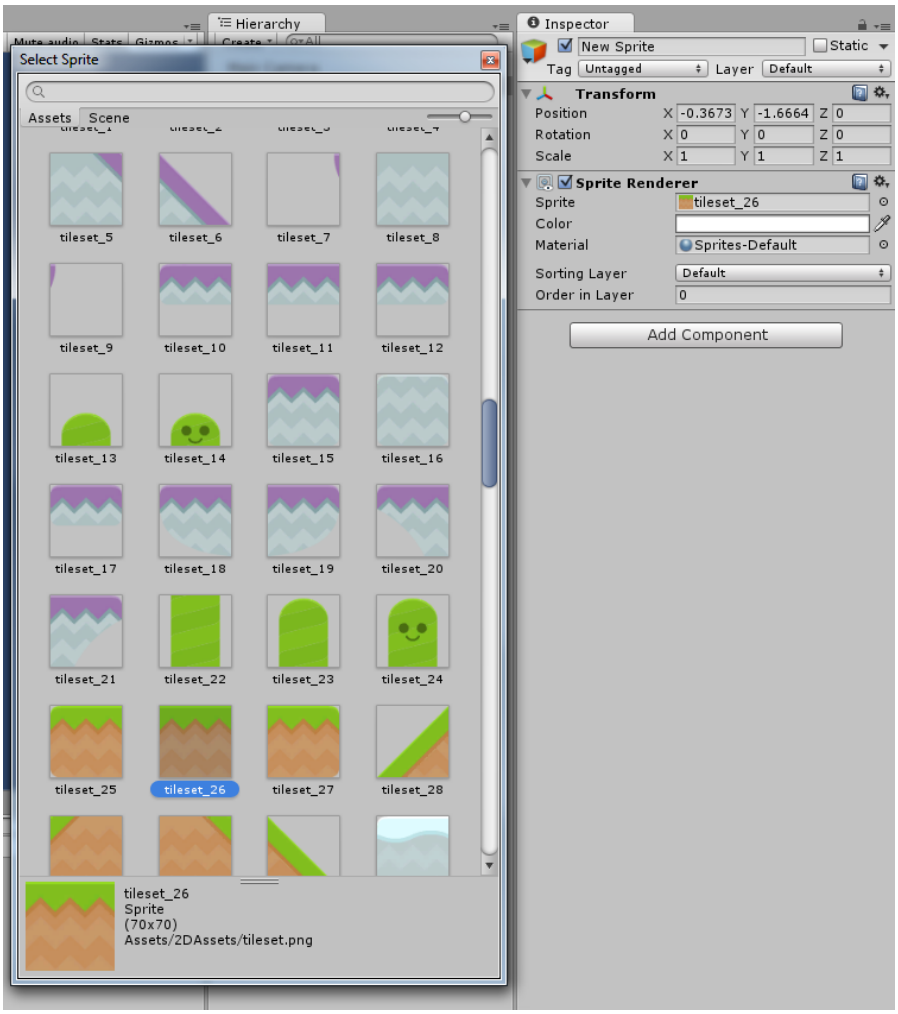
The first thing we're going to want to do is manage our assets. The tileset spritesheet has all the assets for making the level. Click on the spritesheet in the project window, and in the inspector change the sprite mode from single to multiple, and then open the Sprite Editor.



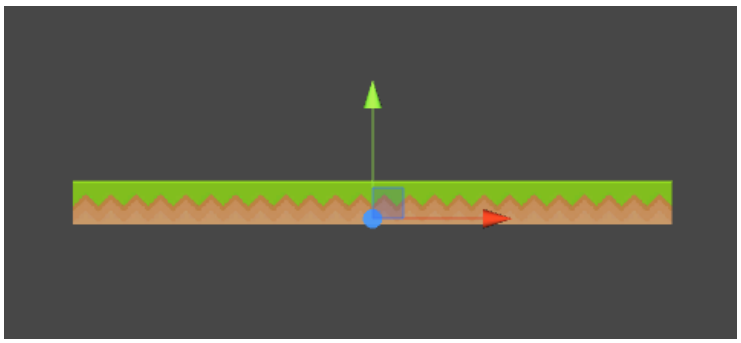
In the sprite editor, click the slice button. Our spritesheet is an even grid so we want to pick grid as the type. The size of each tile is 70 x 70. The Offset and Padding both also need to be 2x2.



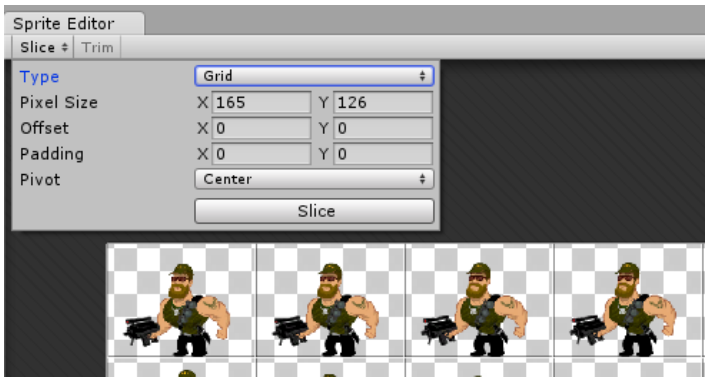
Make sure your settings are correct and click slice. Now each tile is treated as a separate sprite. We can now start adding them to the scene. Under the create menu, select 2D Object->Sprite. This will add an empty sprite to the scene. Select the sprite and in the inspector, under the SpriteRenderer component, click the little circle next to the Sprite field. This will let you select the sprite you want displayed.



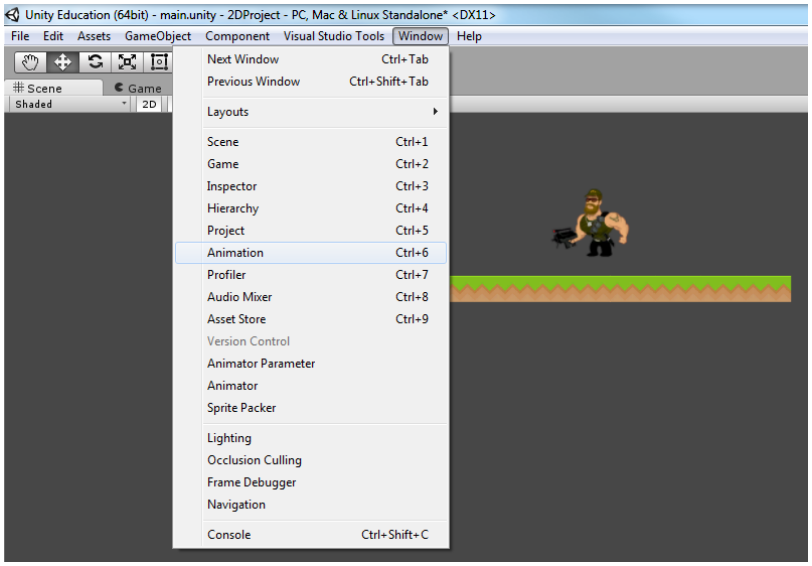
Select one of the ground sprites and we have the first piece of our level. Create some more sprites to build up a small test area. I've just made a simple platform.



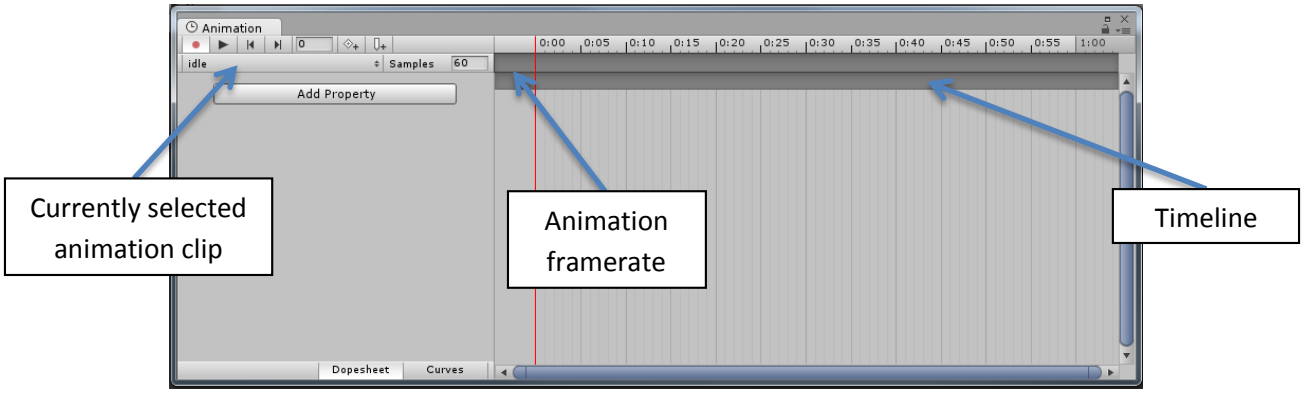
Select all the tiles we've added and add a Box Collider 2D component to all of them. Now we want to create a player character that we can move around our test level. The first thing we want to do is to sort out the animation for our character. Selecting the ChuckNorris spritesheet, we want to set up the frames for our animations. Just like with the tileset, the sprite mode should be set to multiple. Opening up the sprite editor, we want to again use the grid slicing, setting the X and Y to 165 and 126.



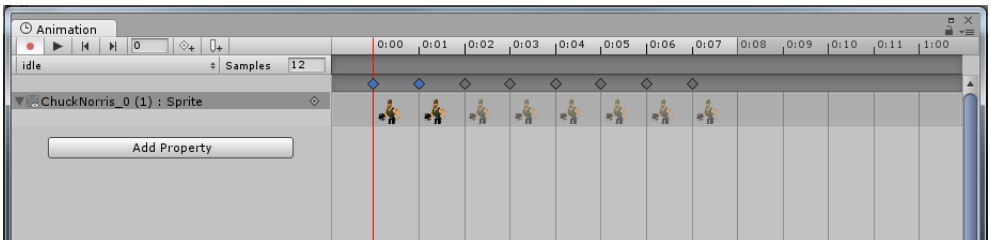
Click and drag the first frame of the Chuck Norris spritesheet into the scene. To create animations for the character, make sure he is selected and open the Animation window through the window menu.



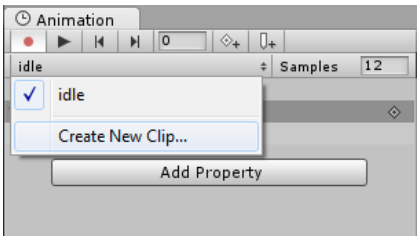
The animation window will prompt you to create a new animation clip and .controller. You should click create and name the first clip "Idle". Now there are a few things we can do in the animation window.



The frame rate of our spritesheet is 12. This is a common framerate for flipbook animation. To make sure the game matches this, set the sample rate to 12. Now drag the first 8 frames from the sprite sheet from the project window into the animation window. Now you should be able to click play and see the animation playing.



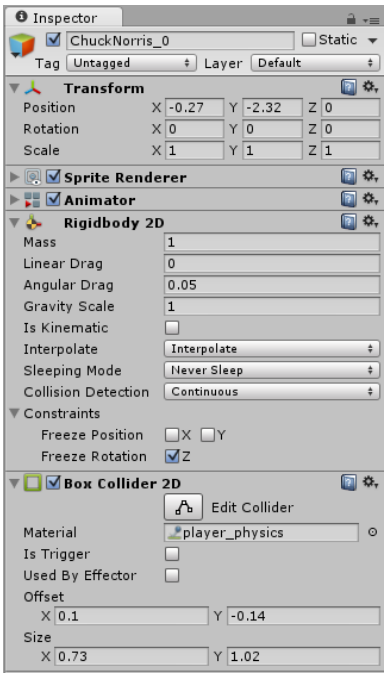
Now click on the dropdown with the animation name and select Create New Clip… This time name it Jump. Again make sure the sample rate is set to 12 and drag the next 4 frames for the jump animation in.



Continue this to make animations for all of the frames in the sprite sheet.

Moving the Player

Now that we have the animations set up, we need to make the code to drive our player. Close the animation window and add a box collider 2D and a rigid body 2D to the player. You'll need to tweak the box collider so it fits the player more nicely. For our player we want the best collision detection available, so set interpolate to interpolate, sleeping mode to never sleep and collision detection to continuous. We also don't want our player rotating, so under constraints, tick Freeze Rotation Z.



We're going to create a new script called PlayerActor to handle movement for our player. Create the script and add it to our player GameObject. These are the variables we're going to add to the top of our PlayerActor script.

```
0 references
public class PlayerActor : MonoBehaviour {

    public float speed;
    public float jump_speed;

    private Rigidbody2D rigid_body;

    private bool grounded = true;
    private float grounded_timer = 0;
```

The speed and jump_speed are how fast the player moves and jumps. These are public so they can be set from the inspector. The rigid_body is a reference to the 2D rigid body on the player. Grounded is a bool telling us if the player is touching the ground or not.

In the start function we just want to grab the rigid body component

```
// Use this for initialization
0 references
void Start () {
    rigid_body = GetComponent<Rigidbody2D>();
}
```

And here's our update function

```
void Update () {
    //Store temp versions of velocity and scale
    Vector2 curr_vel = rigid_body.velocity;
    Vector3 scale = transform.localScale;
    if ( Input.GetKey(KeyCode.A)) {
        //move left
        curr_vel.x = -speed;
        scale.x = 1; //this is so the player is facing the right direction
    } else if (Input.GetKey(KeyCode.D)) {
        //move right
        curr_vel.x = speed;
        scale.x = -1; //this is so the player is facing the right direction
    } else {
        //don't move
        curr_vel.x = 0;
    }
    if ( Input.GetKeyDown(KeyCode.Space) && grounded) {
        //jump!
        curr_vel.y = jump_speed;
        grounded = false;
    }
    transform.localScale = scale;
    rigid_body.velocity = curr_vel;

    //if the collision system doesn't tell us
    //we're on the ground for over 0.2 seconds
    //treat us as not grounded
    grounded_timer += Time.deltaTime;
    if ( grounded_timer > 0.2f ) {
        grounded = false;
    }
}
```

If you notice in the last comment, we're letting the collision system tell us when we're grounded. To do this we need to implement the OnCollisionEnter2D and OnCollisionStay2D.

```
2 references
void CheckCollisionsForGrounded(Collision2D hit)
{
    for (int i = 0; i < hit.contacts.Length; ++i)
    {
        //if the surface we hit is mostly facing up, treat as grounded
        if (Vector2.Dot(hit.contacts[i].normal, Vector2.up) > 0.75f)
        {
            grounded = true;
            grounded_timer = 0;
        }
    }
}

0 references
void OnCollisionEnter2D(Collision2D hit)
{
    CheckCollisionsForGrounded(hit);
}

0 references
void OnCollisionStay2D(Collision2D hit)
{
    CheckCollisionsForGrounded(hit);
}
```

The last thing we need to do is build our mechanim state machine to handle animation transitions properly. We already have an AnimatorController with all the animations in it that was created for us when we created the first animation. Double click on in in the project window. We're going to create two new properties one for the X velocity and one from the Y velocity. Add the following code to your PlayerActor update function.

```
Animator animator = GetComponent<Animator>();
animator.SetFloat("x_velocity", rigid_body.velocity.x);
animator.SetFloat("y_velocity", rigid_body.velocity.y);
```

Exercises

1. Create the transitions for the animations in the AnimatorController
2. Add the ability to shoot to the player. Use the provided bullet spritesheet
3. Find, or create your own enemy spritesheet and make enemies that patrol along platforms and reset the player to his start position if he touches one
4. Replace our test level with a your own creation using the tiles available in the tileset