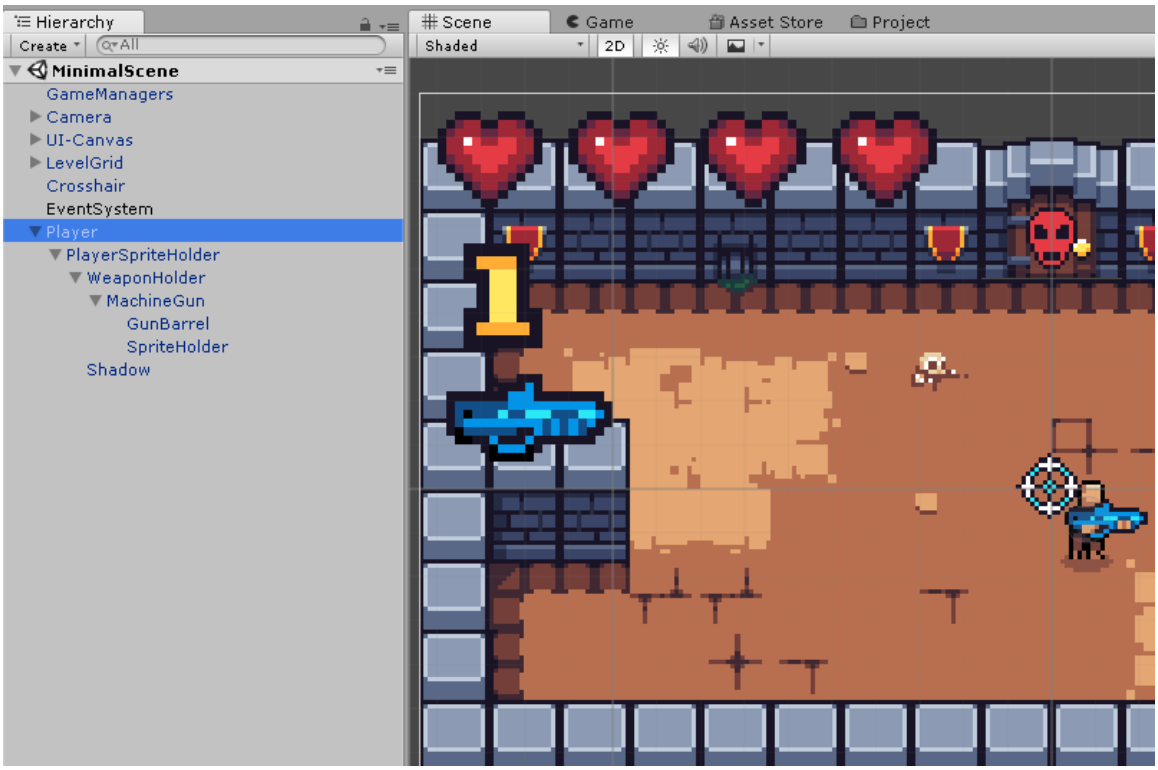
[Documentation]

[CHARACTER CREATION]

Introduction



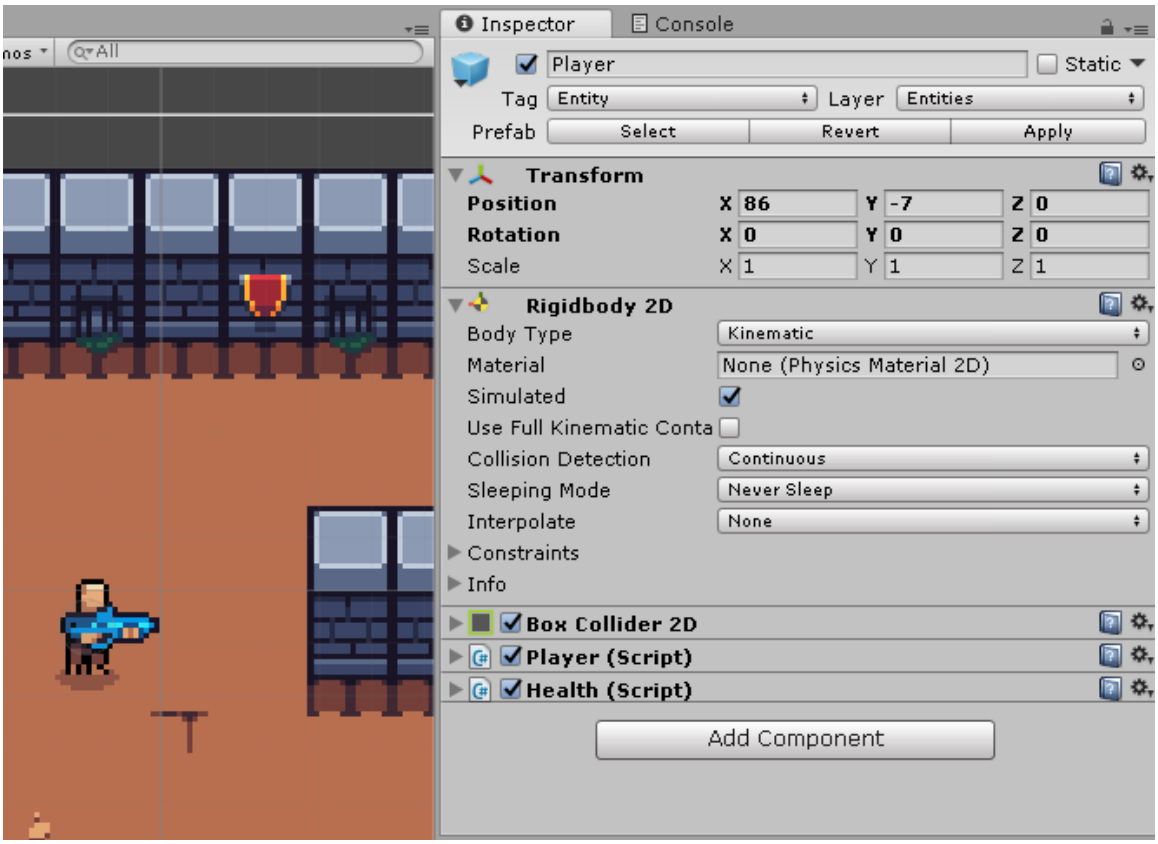
The typical structure of a character, with its sprite holder, weapon holder and weapon.

In the Top Down Shooter Engine “Actor” is a term used to describe any kind of character, whether they’re a player controller characters, enemies, bosses, NPC, etc.

The Actor.cs script is the core class for a Character to work properly, it handles collision checking (AABB) and pixel perfect movement and you’ll need to get familiar with it if you want to expand on later.

Base Concepts

[THE ROOT]

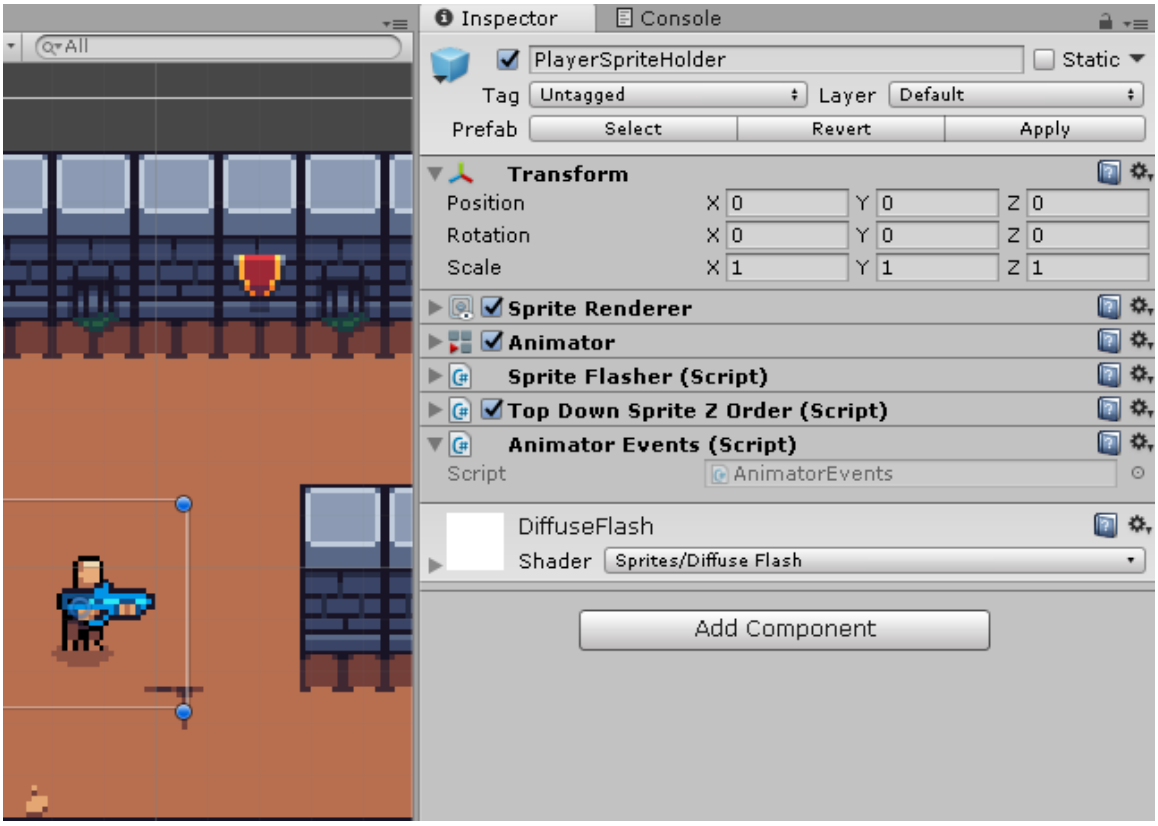


An example of the stack of components commonly found on the root gameobject of a character.

The root gameobject of a character should be set to the Entitie's Layer and usually has these components:

- **BoxCollider2D**: the collider whose size is used to determine collisions.
- **RigidBody2D**: used to provide interaction with standard physics on other objects such as coins, traps, pits, etc. The Rigidbody2D settings should be set like in the picture above.
- **The Actor**: Any character should inherit from the Actor class, some examples of this are the Player, EnemySkeleton and EnemySlime scripts.
- **Health**: The Health component handles damage, health gain/loss, and death.

[THE SPRITE HOLDER]



An example of the stack of components found on sprite holder child gameobject of a character.

The main child gameobject of a character should it's sprite holder and usually has these components:

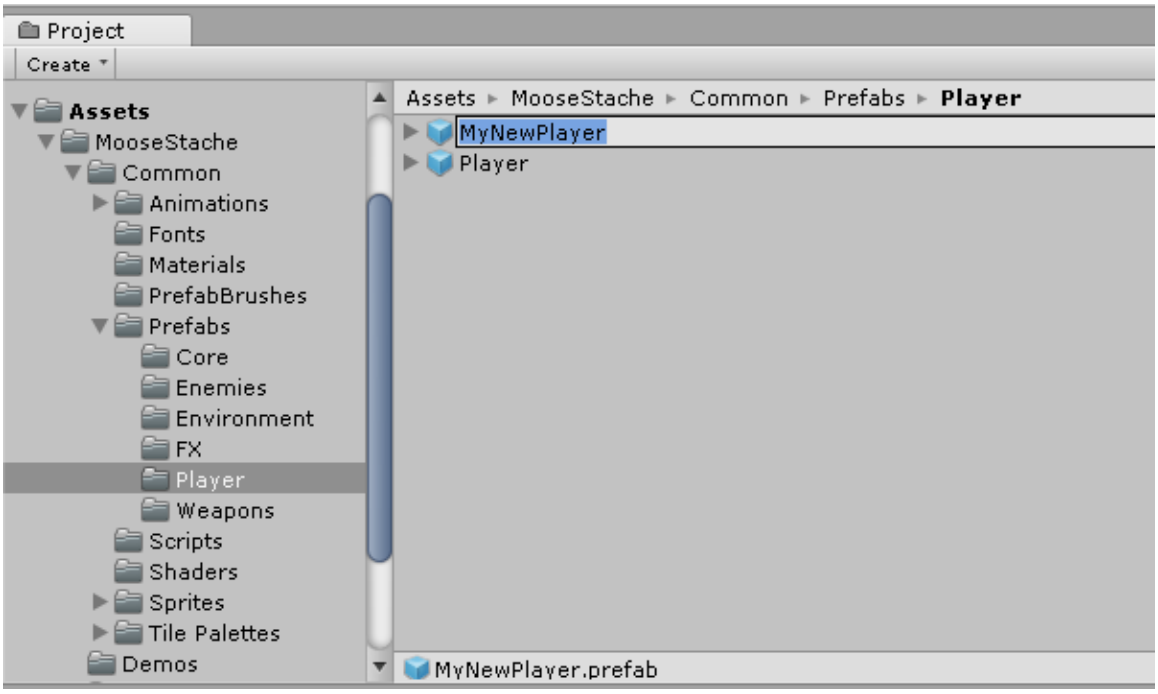
- **SpriteRenderer**: Displays the actual character sprites.
- **Animator**: Used to hold, reference and play the character animations.
- **SpriteFlasher**: Not mandatory, this component is used to flash sprites White on certain events, like taking damage. Also it requires the sprite renderer to have a material with the Diffuse Flash Shader (included in the asset).
- **TopDownSpriteZOrder**: This handles the order in layer of the object base don it's Y position to handle ordering of overlapping 2d objects/sprites.
- **AnimatorEvents:** This is used to store methods which will be called using animation events.

How to Create a new Character

There is several ways you can create a new AI or playable character. We'll cover the 2 recommended ones:

[Copy Approach]

The fastest and easiest way to create a new character is to find one you like in the demos or prefabs, and create yours from that. The process for that is quite simple:



Copying the existing Player prefab.

1. **Find a character/actor you like** in one of the demos or prefabs.
2. **Locate its prefab** (select the Character in Scene View, and in its inspector, at the very top right under the prefab's name and tag there's a Select button).
3. **Duplicate the prefab** (cmd + D).
4. **Rename it** to whatever you want your Character to be called.
5. **Make the changes you want.** Maybe you'll just want to replace some settings, maybe you'll want to change the sprite and animations. It's up to you from there.

[Components Approach]

You can also create a character from scratch, and you can find reference images of it on “The Root” and “The Sprite Holder” sections, the process is:

1. **Create an empty gameobject** this will be your root. Ideally you’ll want to separate the Character part from the visual part, as shown above. The best possible hierarchy has the Actor/RigidBody2D/BoxCollider2D/Health on the Root, and then nests the visual parts (sprite, model, animations, animator events, etc).
2. At the top of the inspector, **set the tag to Entity** and **set the layer to Entities**.
3. Add a **BoxCollider2D**. Adjust its size to match your sprite/model dimensions and check **Is Trigger**.
4. Add a **RigidBody2D**. Set its type to kinematic, collision detection to continous and sleeping mode to never sleep.
5. Add the **Player** script if it’s a playable character or a custom script which inherits from **Actor** to make your own custom characters. For more Info on the Actor class read the “Introduction” and “The Root” sections.
6. **Create an empty gameobject** and make it a child of your root gameobject. This will be your sprite holder / visual parts holder.

[Contact Us]

[Talk to us on Discord]
[Email: Moose_Stache@hotmail.com]

Disclaimer: This is part of the documentation for the Pixel Top-Down Engine made by MooseStache.