

# Physics Puzzle Platformer

## Documentation



**The Physics Puzzle Platformer is A template or base framework for creating active physics games much like the popular titles Gangbeasts, Human Fall Flat etc.**

In this documentation we'll do A walkthrough of how the template itself funtions which will be a blueprint for creating new levels and then we take A look at how active physics work.

**First let's start with dissecting the project by explaining how things work within the template framework.**

After importing the package into a fresh project, you will find 2 scenes (Menu and Game). We'll be focusing on the Game scene as this is where all the physics puzzle platformer and active physics ragdoll stuff are constructed.

### **[Game Scene]**



Inside the Game scene we have a few important objects, layers and tags.

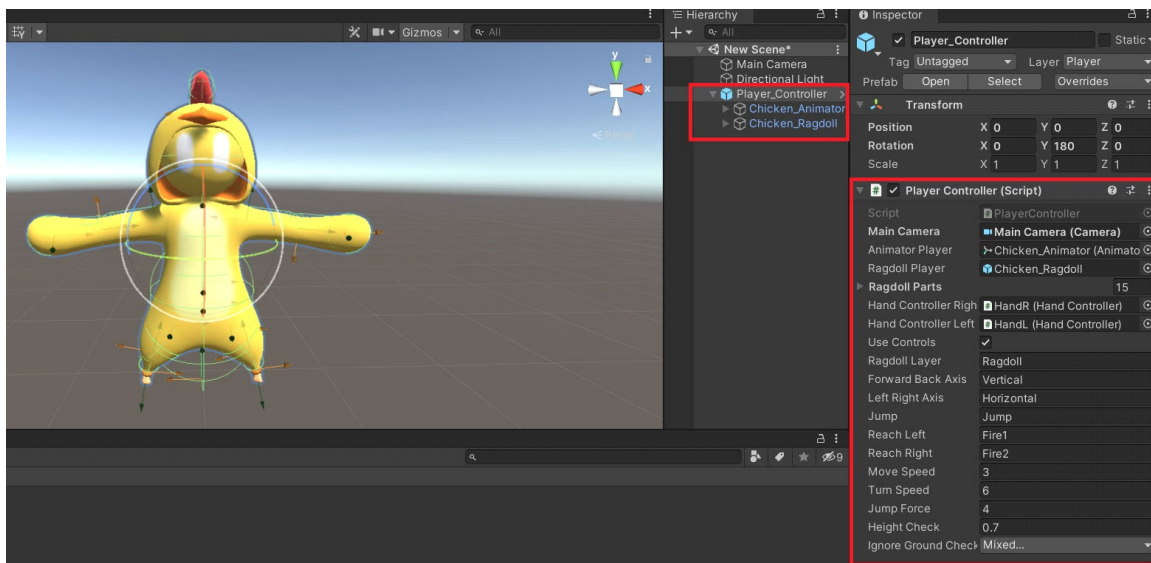
One object in particular would be the **Player\_Controller** which consists 3 objects.

1) Main object with the controller script.

2) A simple rigged model that can be animated (*this object is visually hidden*)

3) The same rigged model but with physics joints, colliders, additional scripts etc.

(*This object mimics the animated object with joint rotations but keeps physics interaction*)

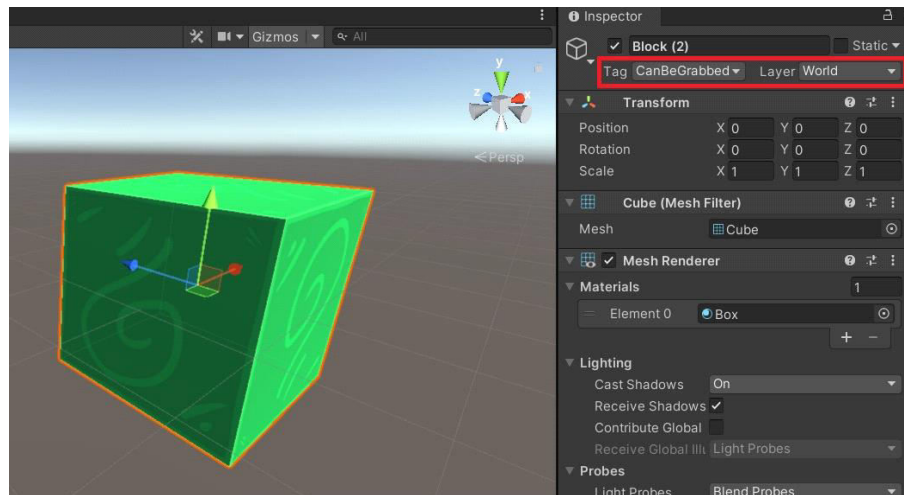


**Another important type of object is the grabbable, pickup physics objects.**

**Here we have a cube object with 3 requirements for it to be interactable by the Player\_Controller.**

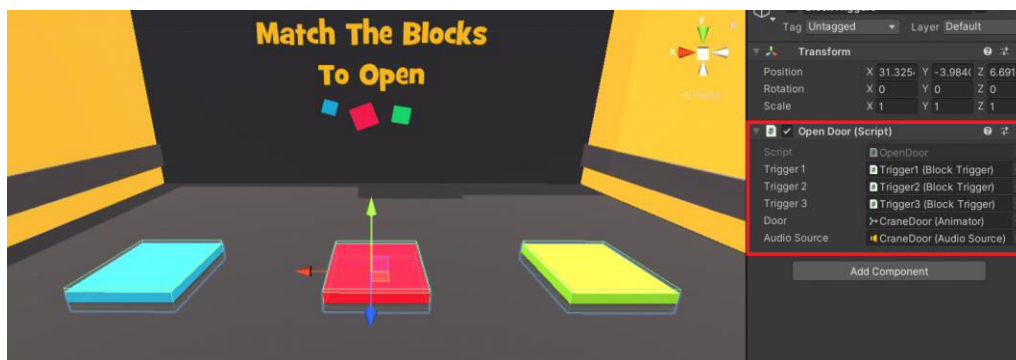
- 1) A rigidbody
- 2) "CanBeGrabbed" tag
- 3) "World" Layer

This will make the object interact with the environment as well as be possible for the Player\_Controller to grab/pickup the object by creating a fixed joint from the player's hand connected to the object.

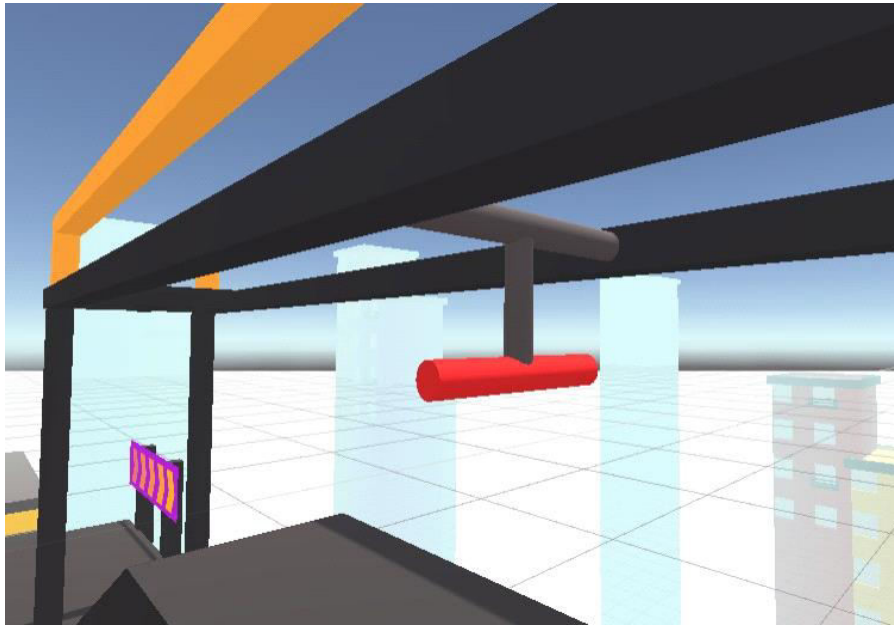


**In this puzzle we use our physics objects and player controller to pickup and place the respective blocks to open a door.**

The door simply plays an animation along A sound when all 3 triggers are true.

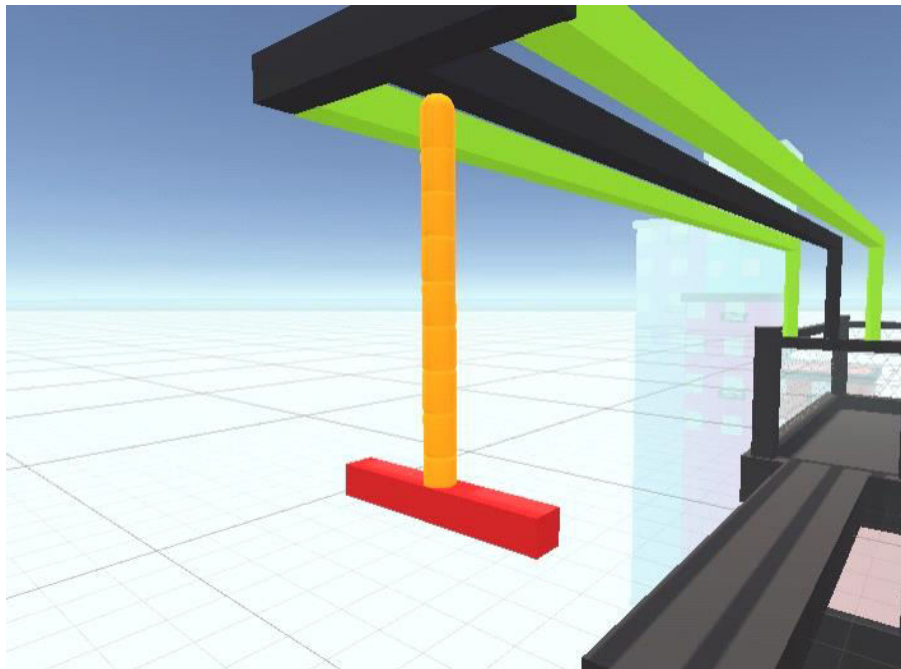


**Zipline and rope swing objects works the same way, with the correct tag, layer and rigidbody we are able to grab them.**



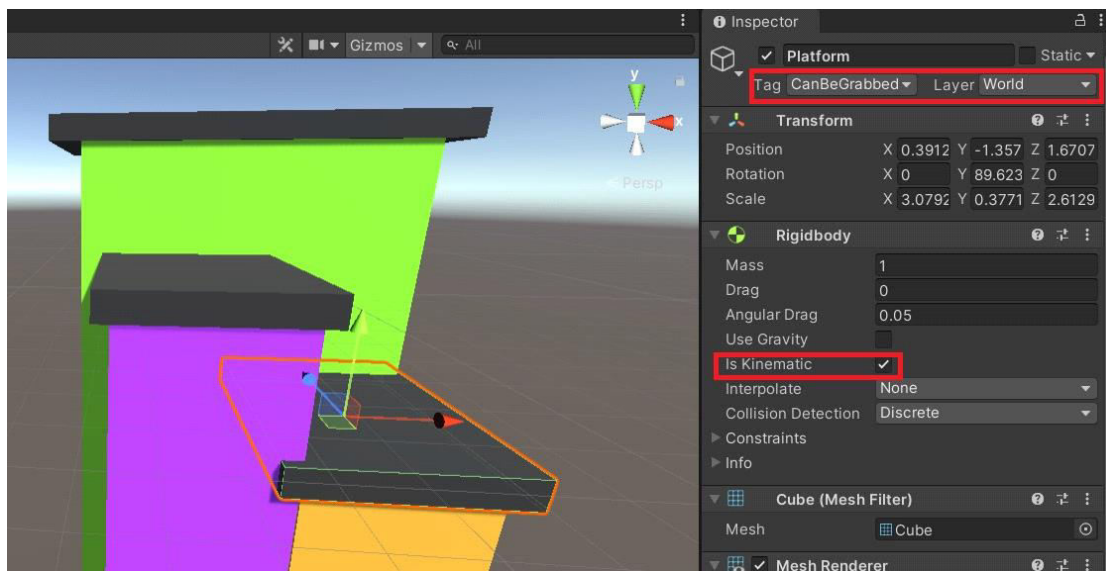
**(Rope swing)**

**Tag "CanBeGrabbed", Layer "World" and contains A rigidbody.**



**There are climbable platforms which require the same tag and layer as pickup objects but its rigidbody is set to Kinematic.**

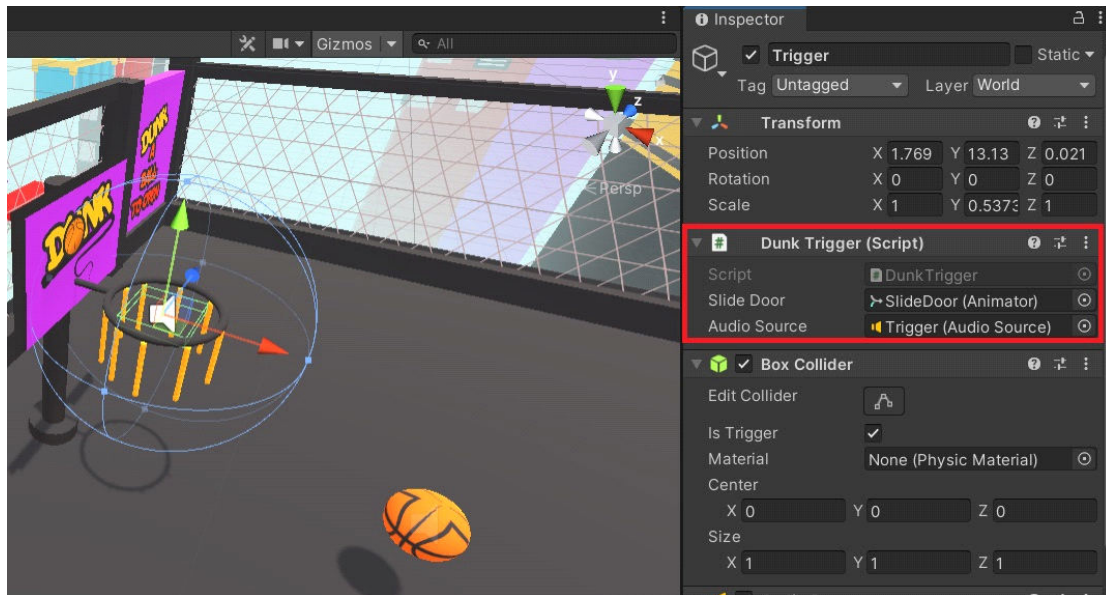
The Player\_Controller's grabbing will check if the object is Kinematic, if so and both hands are jointed to the object, A climbing action will be performed after pressing jump and A raycast check from the head detects no obstacles.



**Other interactable objects are simply trigger checks.**

for example the basketball section checks if an object on the layer "Ball" passes through the trigger inside the hoop then play A sound and an animation on the slide door.

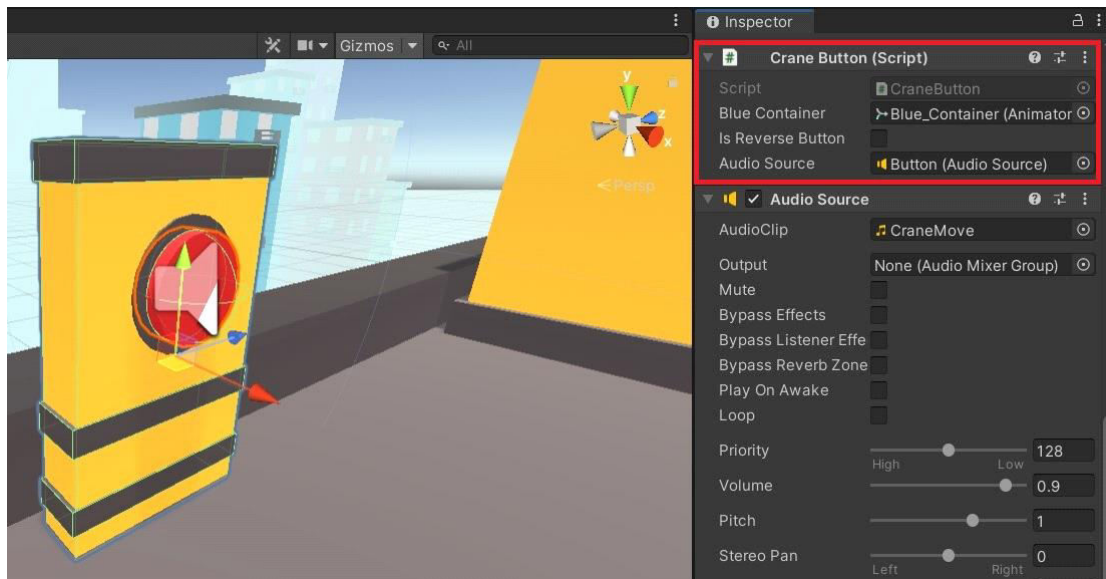
**(The ball also uses the same tag "CanBeGrabbed", layer "Ball" and A rigidbody)**





## More interactable objects

The Buttons also have a trigger collider and if the Player\_Controller's hand grabs, as it enters the trigger we perform some actions with a script, in this instance we play a sound and play an animation on the Blue Container object.

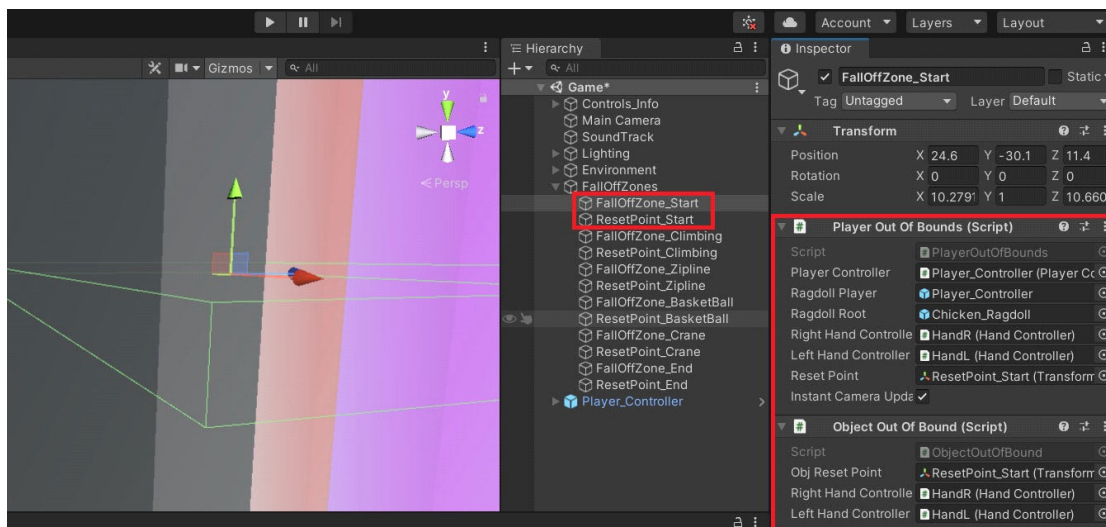




### Fall off zones also works with A trigger.

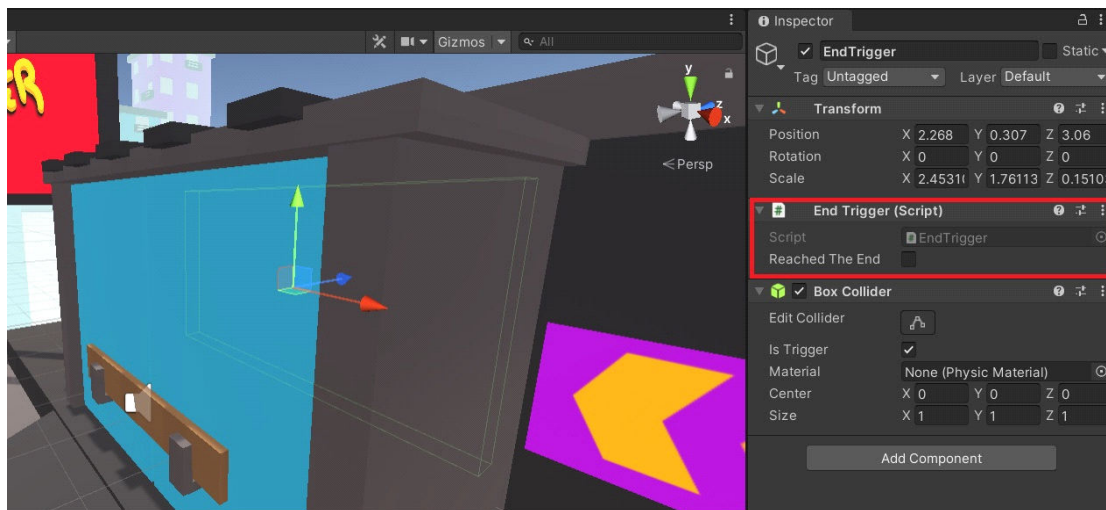
The trigger contains 2 scripts, to move the Player\_Controller to the reset point and another to move other objects if they end up falling off. The difference in these two scripts are simply the joints handling, breaking grabbed object joints from hands etc. to avoid glitches when instantly moving the Player.

The fall off script for the Player\_Controller checks the layer to differentiate it from generic physics objects to deal with those hickups we mentioned earlier.



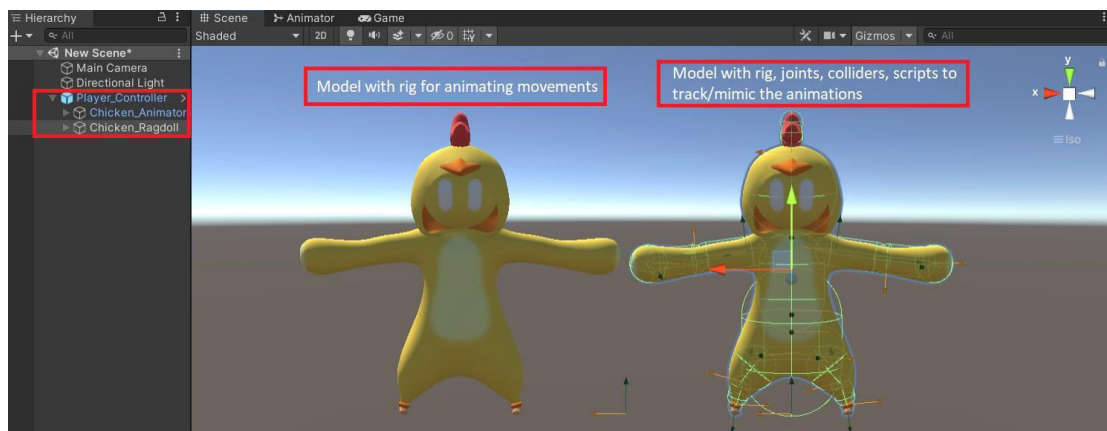
### The end zone uses the same approach.

When the Player\_Controller enters the trigger we check for the layer "Player" and if so we set some values, we then check those values after the Player\_Controller falls through the last fall off zone, if we passed the end zone check and fall off we execute a screen fade, load the menu scene after that.

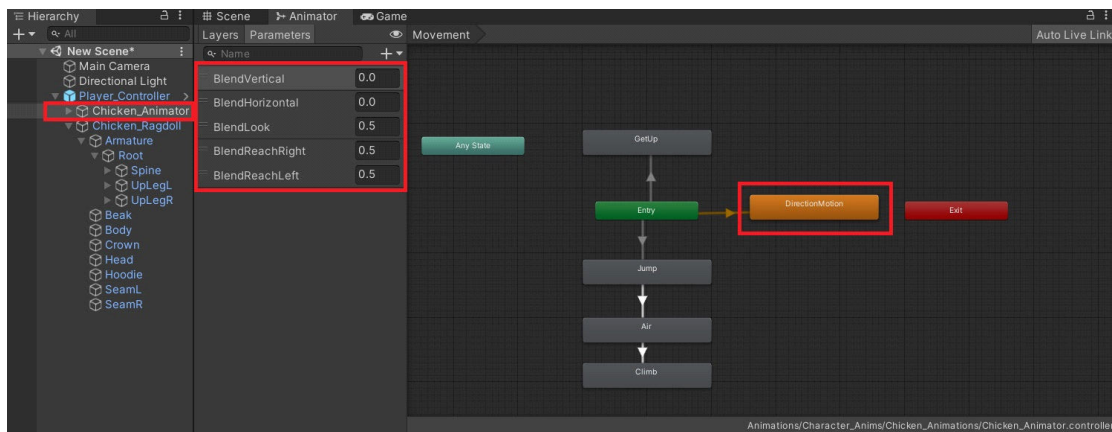


**Now that we have summarized the template side of things, lets dive into what makes this style of game what it is, the active physics ragdoll.**

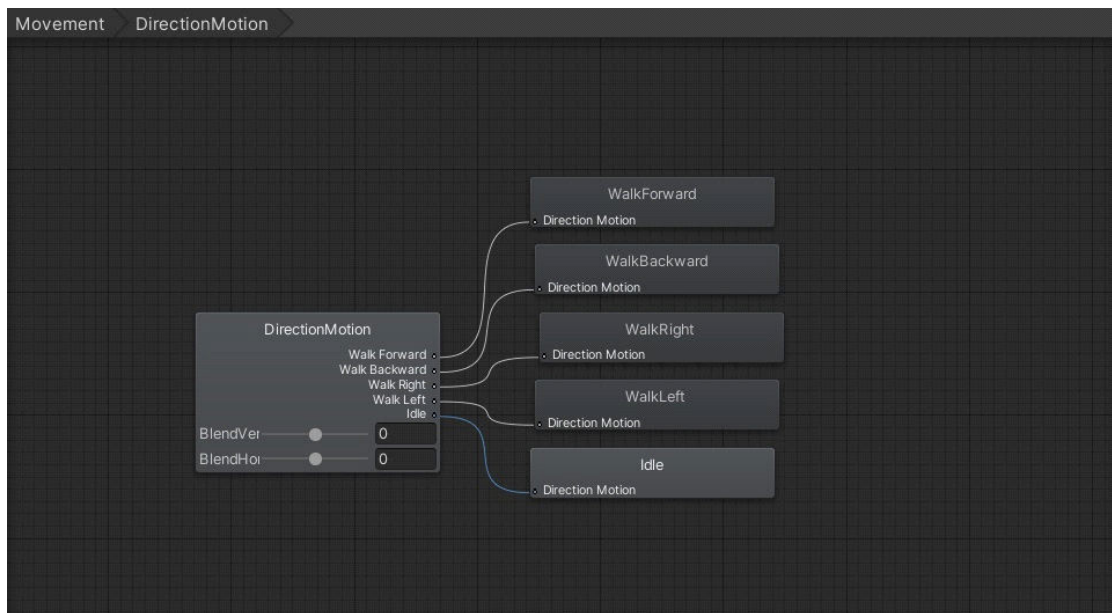
To stress the importance of the following approach is fairly simple, we have 2 of the same models that are rigged, we hide the mesh on the one we use to animate our character and then we have A second one with joints, joint drive and target rotations, the target rotation values will be thus of our animated model, by doing so we can animate our player and have it still interact with the world physically.



Our animated model uses blend trees, layers and parameters to blend between animations such as reach with the arms and bending of the spine, basically overriding some parts of the animation.



We do the same with our basic movement to smoothly transition between animations.



This is how our 2 models work together, we have the animated model and the ragdoll model that have a script on each individual bone that reference which transform to keep track of which is the same bone on the animated model.



This script keeps track of our animated model spine and then uses those transform values, convert it into joint space and then applies it to the ragdoll's spine joint target rotations. The "Pose" are forced by the joint drive.





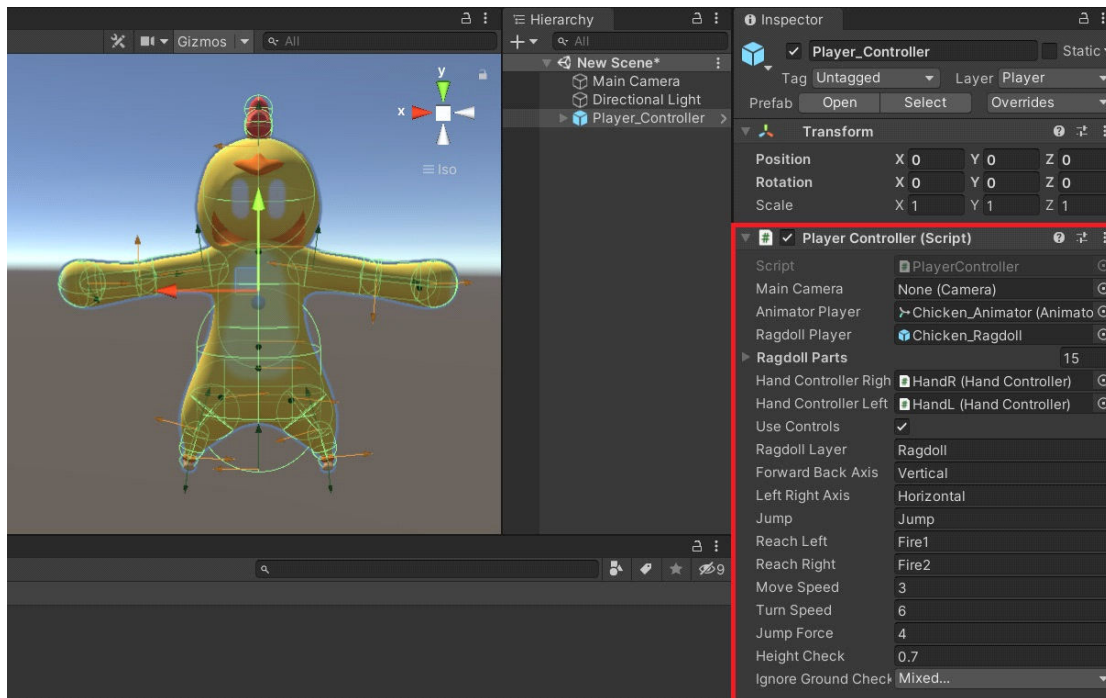
**Lets talk about the active ragdoll itself, besides the animation tracking we have some important aspects to cover.**

As mentioned the active ragdoll consists out of joints, colliders, scripts. To simplify the concept, we turn our model into a regular ragdoll using configurable joints on all our rig bones, construct a working physics ragdoll that would plop on the ground with gravity and then add joint drive to all those bone joints to stiffen it. We then have a capsule with the same components which we will use to attach the ragdoll to with another joint.

The capsule we use as the main object to drive movement whilst we perform animation tracking on the ragdoll.

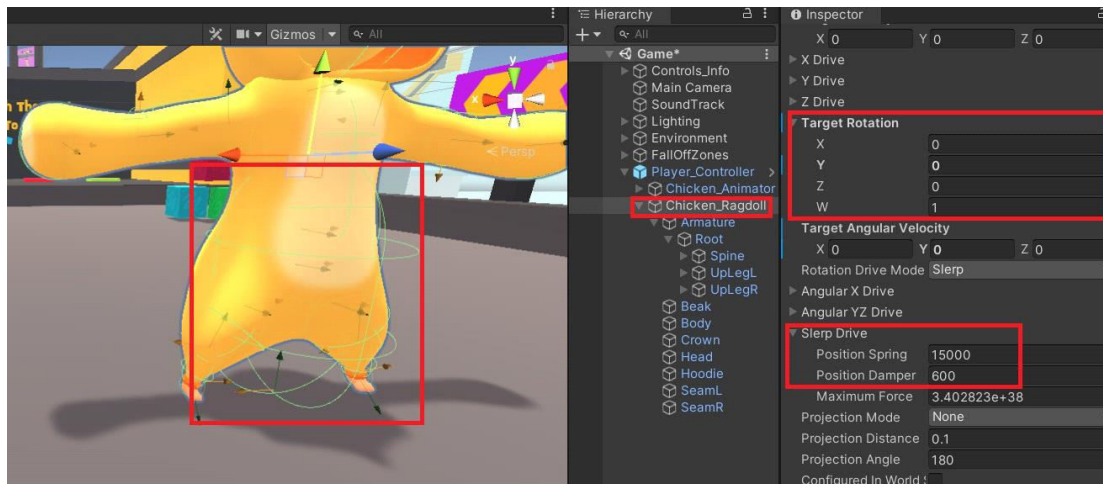
For our capsule object we also have a configurable joint that connects to just a point in space and joint drive to keep it upright therefore also keeps our ragdoll standing.

Our container object called "Player\_Controller" contains the controller script that moves, play animations, checks grounded, listens to inputs etc. applying all needed actions to our animated model and ragdoll.





This is where we attached our capsule, using A configurable joint we set the drive high, using the target rotation to keep it from falling over and then also useful for rotating the player.



(PlayerController script)

In our PlayerController script we check ground with A raycast, setting joint drive accordingly

```

160 //---Player Grounded
161 void PlayerGrounded()
162 {
163     Ray ray = new Ray (ragdollPlayer.transform.position, -Vector3.up);
164     RaycastHit hit;
165
166     //Balance when ground is detected
167     if (Physics.Raycast(ray, out hit, HeightCheck, ~ignoreGroundCheckOn) && inAir && !ragdollMode)
168     {
169         inAir = false;
170         physicsJoint.slerpDrive = DriveOnController;
171     }

```

we move the capsule with velocity

```

353 //---Player Movement
354 void PlayerMovement()
355 {
356     Direction = ragdollPlayer.transform.rotation * new Vector3(Input.GetAxisRaw(leftRightAxis), 0.0f, Input.GetAxisRaw(forwardBackAxis));
357     Direction.y = 0f;
358     physicsBody.velocity = Vector3.Lerp(physicsBody.velocity, (Direction * moveSpeed) + new Vector3(0, physicsBody.velocity.y, 0), 0.8f);
359 }

```

we use the configurable joint's target rotation and set it to our camera direction to rotate our player

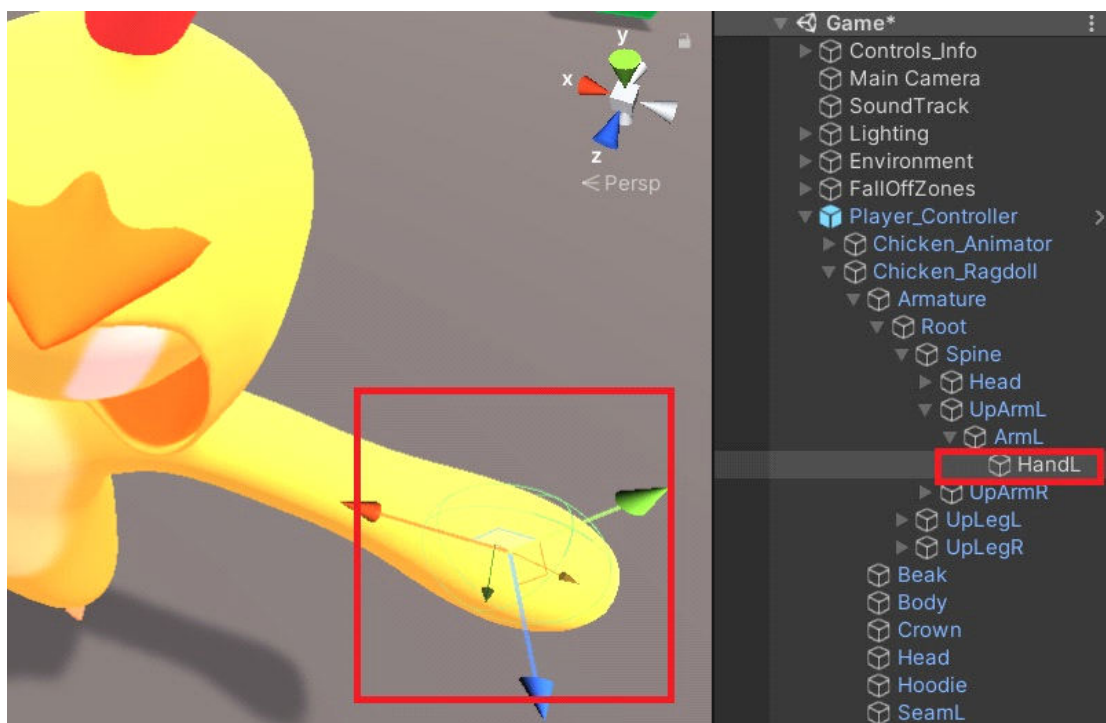
```

488 //---Player Rotation
489 void PlayerRotation()
490 {
491     var lookPos = mainCamera.transform.forward;
492     lookPos.y = 0;
493     var rotation = Quaternion.LookRotation(lookPos);
494     physicsJoint.targetRotation = Quaternion.Slerp(physicsJoint.targetRotation, Quaternion.Inverse(rotation), Time.deltaTime * turnSpeed);
495 }

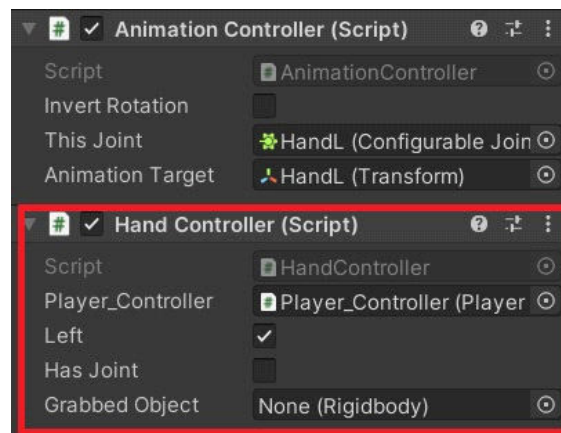
```

**For the ragdoll hands we have A script for handling the grabbing.**

Much like the capsule we have all the body parts on joint drive, the hands in particular holds the script that deals with grabbing physics objects when player input is active, in cases where we press the left mouse button down, our animated model will play A "reachLeft" animation, we send those animated transform values to our ragdoll joint target rotation, when this is happening our hand script is listening.



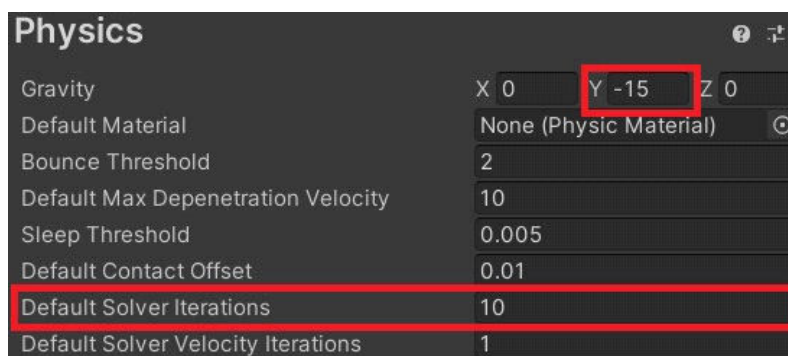
Each hand have A script to deal with grabbing individually



When player input is detected, inside this script we check for A rigidbody and the tag "CanBeGrabbed" on the object collided with then create a fixed joint connecting between the hand and the object.

```
68 //Grab on collision
69 void OnCollisionEnter(Collision col)
70 {
71     if(Player_Controller.useControls)
72     {
73         //Left Hand
74         //On mouse pressed and collides, create joint
75         if(isLeft)
76         {
77             if(col.gameObject.tag == "CanBeGrabbed" && col.gameObject.layer != LayerMask.NameToLayer(Player_Controller.ragdollLayer) && !hasJoint)
78             {
79                 if(Input.GetAxisRaw(Player_Controller.reachLeft) != 0 && !hasJoint)
80                 {
81                     hasJoint = true;
82                     GrabbedObject = col.gameObject.GetComponent<Rigidbody>();
83                     this.gameObject.AddComponent<FixedJoint>();
84                     this.gameObject.GetComponent<FixedJoint>().breakForce = Mathf.Infinity;
85                     this.gameObject.GetComponent<FixedJoint>().connectedBody = col.gameObject.GetComponent<Rigidbody>();
86                 }
87             }
88         }
89     }
90 }
```

The project settings also have A big impact, we bumped up the gravity A bit and set the solver iterations higher to compliment the physics joints.



**That covers the fundamentals of our Physics Puzzle Platformer template, to understand more about the code, we have added comments inside the scripts to help indicate what each section is for.**

**If you have any reasonable questions then feel free to contact TFM support:**

**`thefamousmouse.developer@gmail.com`**

***Thank you for supporting us and goodluck with your project!***