DODGE AND RUN 3D

Documentation

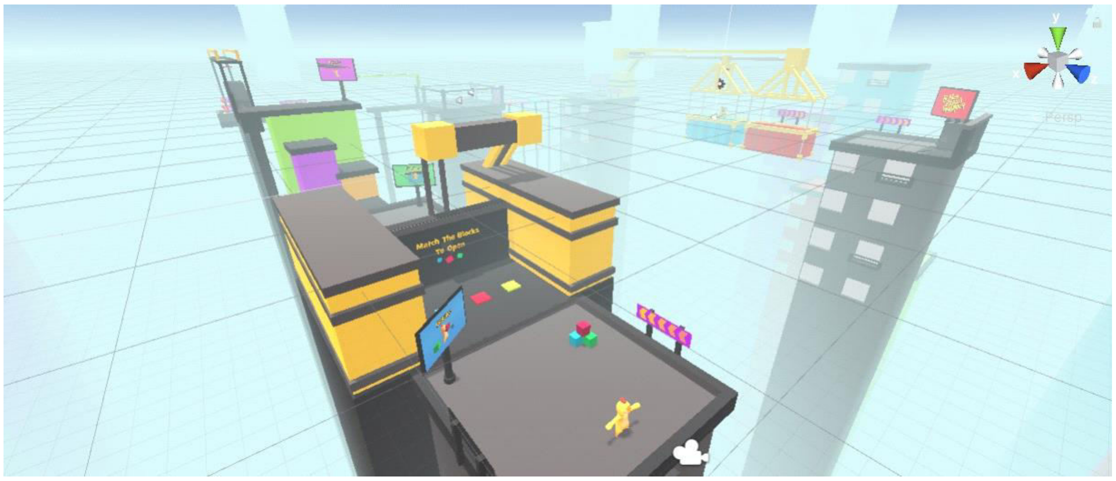
The Dodge and Run 3D is base framework for creating active physics games much like the popular titles Gangbeasts, Human Fall Flat etc.

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.

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.

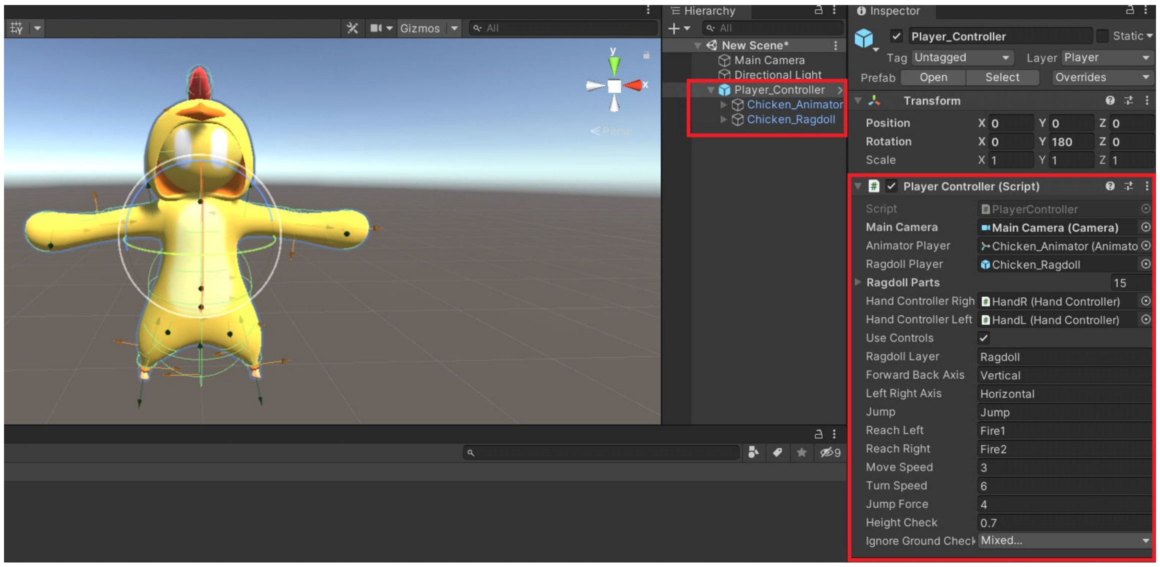
One object in particular would be the Player_Controller which consists 3 objects.

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)



New Scene*
- Main Camera
- Directional Light
- Player_Controller
  - Chicken_Animator
  - Chicken_Ragdoll

Player_Controller — Static
Tag: Untagged — Layer: Player
Prefab: Open | Select | Overrides

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	180	0
Scale	1	1	1

Player Controller (Script)

Script	PlayerController
Main Camera	Main Camera (Camera)
Animator Player	Chicken_Animator (Animato
Ragdoll Player	Chicken_Ragdoll
Ragdoll Parts	15
Hand Controller Righ	HandR (Hand Controller)
Hand Controller Left	HandL (Hand Controller)
Use Controls	✓
Ragdoll Layer	Ragdoll
Forward Back Axis	Vertical
Left Right Axis	Horizontal
Jump	Jump
Reach Left	Fire1
Reach Right	Fire2
Move Speed	3
Turn Speed	6
Jump Force	4
Height Check	0.7
Ignore Ground Check	Mixed...

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

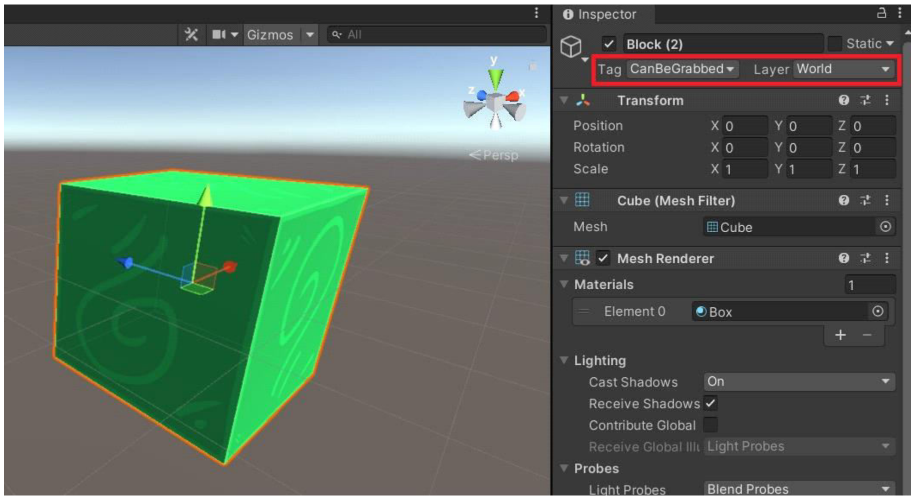
Here we have a cube object with 3 requirements for it to be interactable by the Player_Controller.

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.



Block (2) Static

Tag CanBeGrabbed Layer World

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Cube (Mesh Filter)

Mesh Cube

Mesh Renderer

Materials 1

Element 0 Box

Lighting

Cast Shadows On

Receive Shadows ✓

Contribute Global

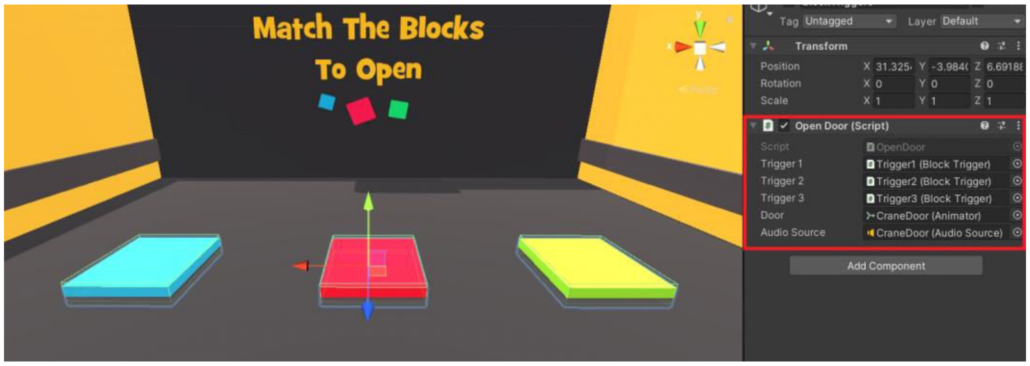
Receive Global Illu Light Probes

Probes

Light Probes Blend Probes

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.



Tag Untagged Layer Default

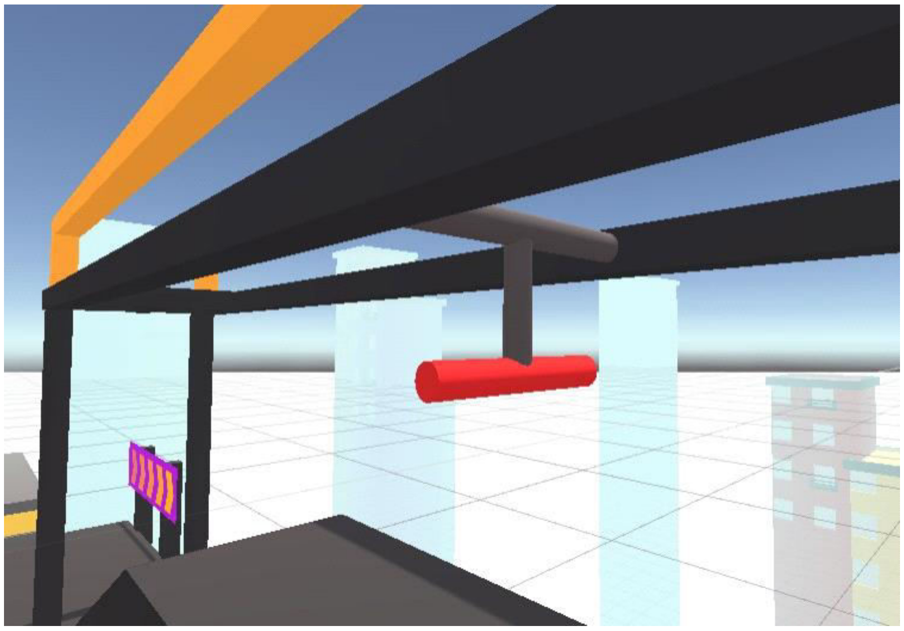
Transform

Position	X 31.325	Y -3.984	Z 6.6918
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Open Door (Script)

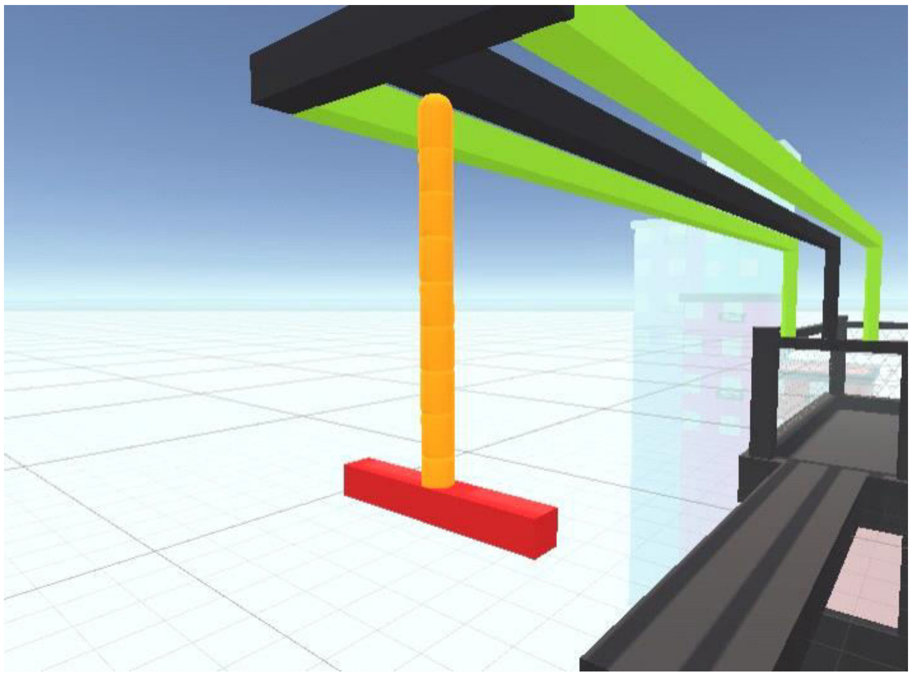
Script	OpenDoor
Trigger 1	Trigger1 (Block Trigger)
Trigger 2	Trigger2 (Block Trigger)
Trigger 3	Trigger3 (Block Trigger)
Door	CraneDoor (Animator)
Audio Source	CraneDoor (Audio Source)

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



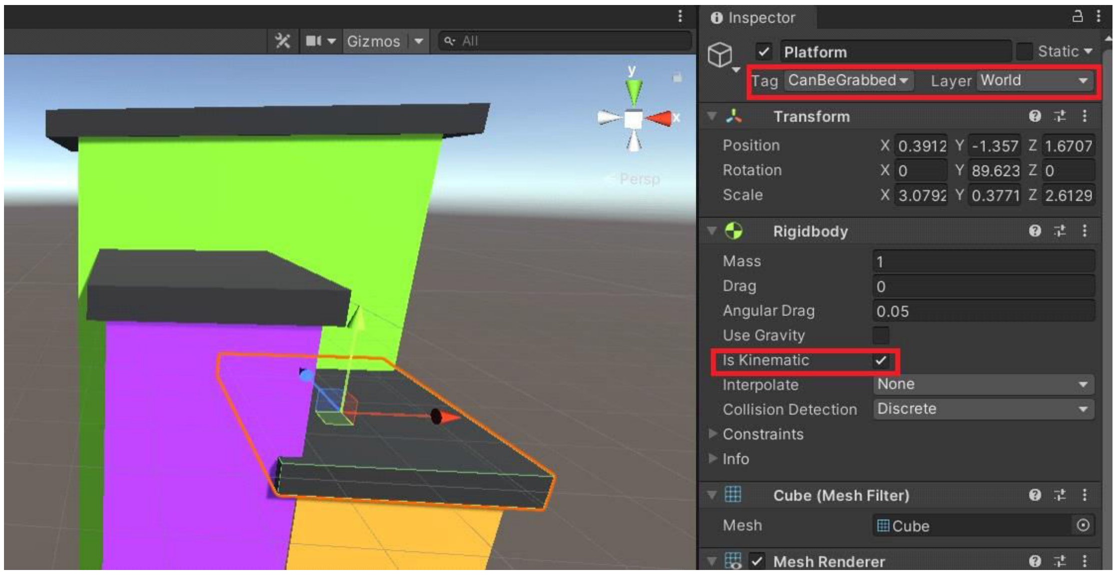
(Rope swing)

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.



Platform ✓ Static
Tag CanBeGrabbed Layer World

Transform
Position X 0.3912 Y -1.357 Z 1.6707
Rotation X 0 Y 89.623 Z 0
Scale X 3.0792 Y 0.3771 Z 2.6129

Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic ✓
Interpolate None
Collision Detection Discrete
Constraints
Info

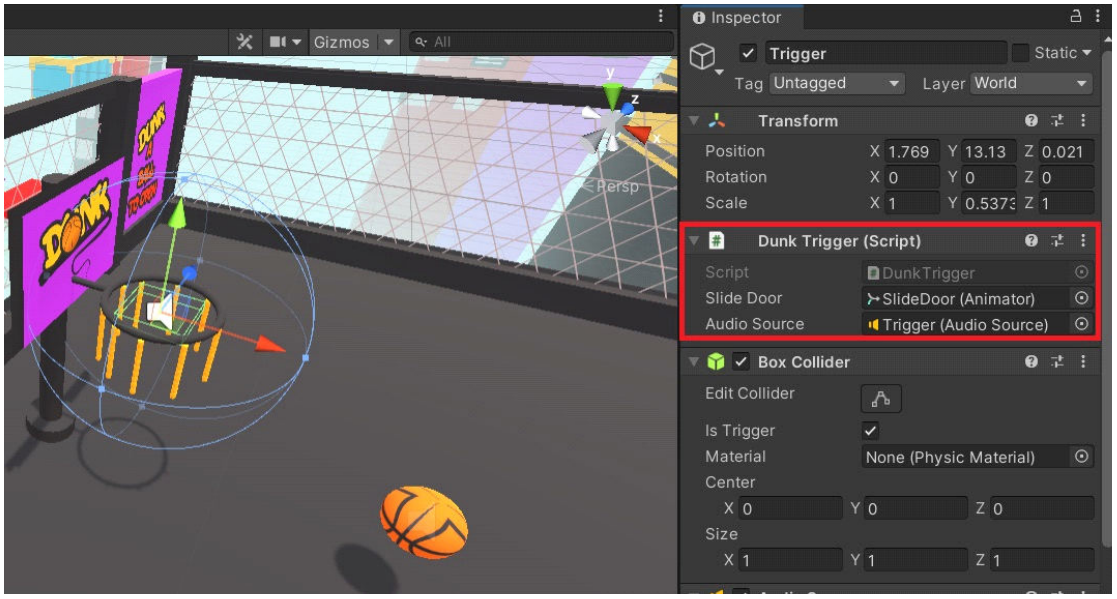
Cube (Mesh Filter)
Mesh Cube

✓ Mesh Renderer

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.

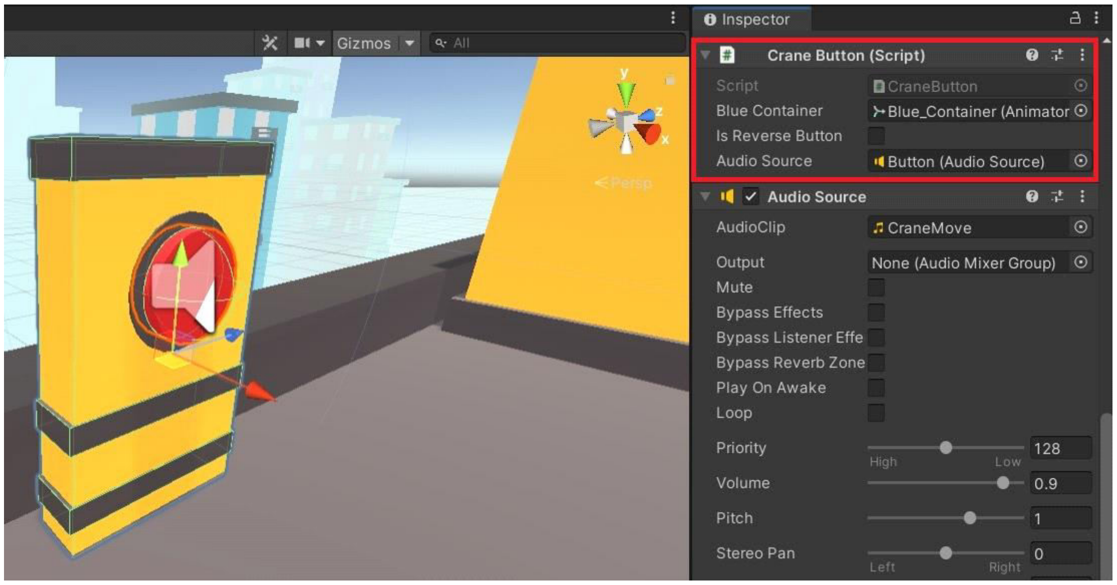
(The ball also uses the same tag "CanBeGrabbed", layer "Ball"   and A rigidbody)



```
Trigger                                Static
Tag  Untagged          Layer  World

Transform
Position    X 1.769    Y 13.13    Z 0.021
Rotation    X 0        Y 0        Z 0
Scale       X 1        Y 0.5373   Z 1

Dunk Trigger (Script)
Script          DunkTrigger
Slide Door      SlideDoor (Animator)
Audio Source    Trigger (Audio Source)

Box Collider
Edit Collider
Is Trigger      ✓
Material        None (Physic Material)
Center
   X 0     Y 0     Z 0
Size
   X 1     Y 1     Z 1
```

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.



Crane Button (Script)

Script	CraneButton
Blue Container	Blue_Container (Animator
Is Reverse Button	
Audio Source	Button (Audio Source)

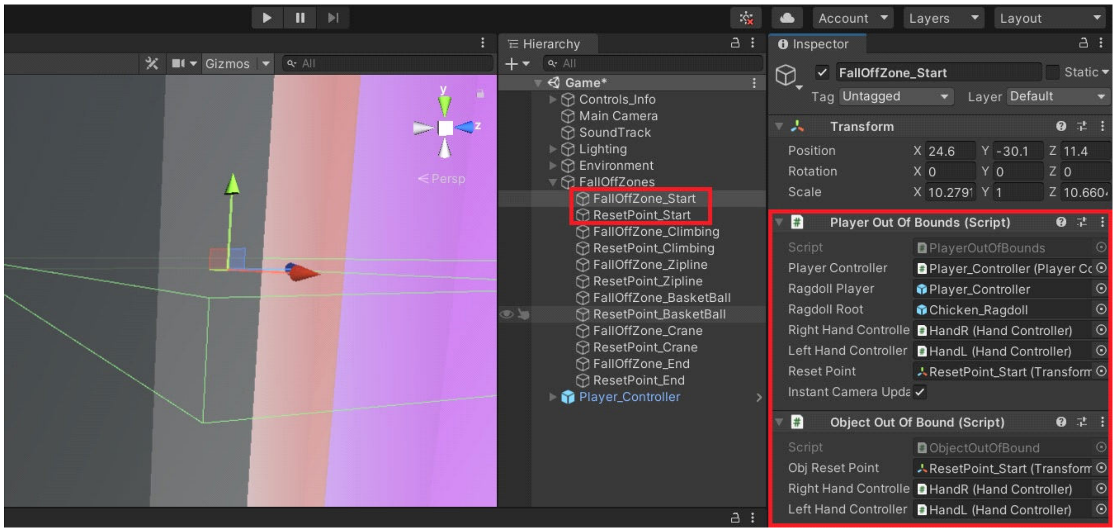
Audio Source

AudioClip	CraneMove
Output	None (Audio Mixer Group)
Mute	
Bypass Effects	
Bypass Listener Effe	
Bypass Reverb Zone	
Play On Awake	
Loop	
Priority (High – Low)	128
Volume	0.9
Pitch	1
Stereo Pan (Left – Right)	0

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 handeling, 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 differenciate it from generic physics objects to deal with those hickups we mentioned earlier.



Game*
- Controls_Info
- Main Camera
- SoundTrack
- Lighting
- Environment
- FallOffZones
  - FallOffZone_Start
  - ResetPoint_Start
  - FallOffZone_Climbing
  - ResetPoint_Climbing
  - FallOffZone_Zipline
  - ResetPoint_Zipline
  - FallOffZone_BasketBall
  - ResetPoint_BasketBall
  - FallOffZone_Crane
  - ResetPoint_Crane
  - FallOffZone_End
  - ResetPoint_End
- Player_Controller

FallOffZone_Start — Static
Tag Untagged — Layer Default

Transform

	X	Y	Z
Position	24.6	-30.1	11.4
Rotation	0	0	0
Scale	10.2791	1	10.660

Player Out Of Bounds (Script)

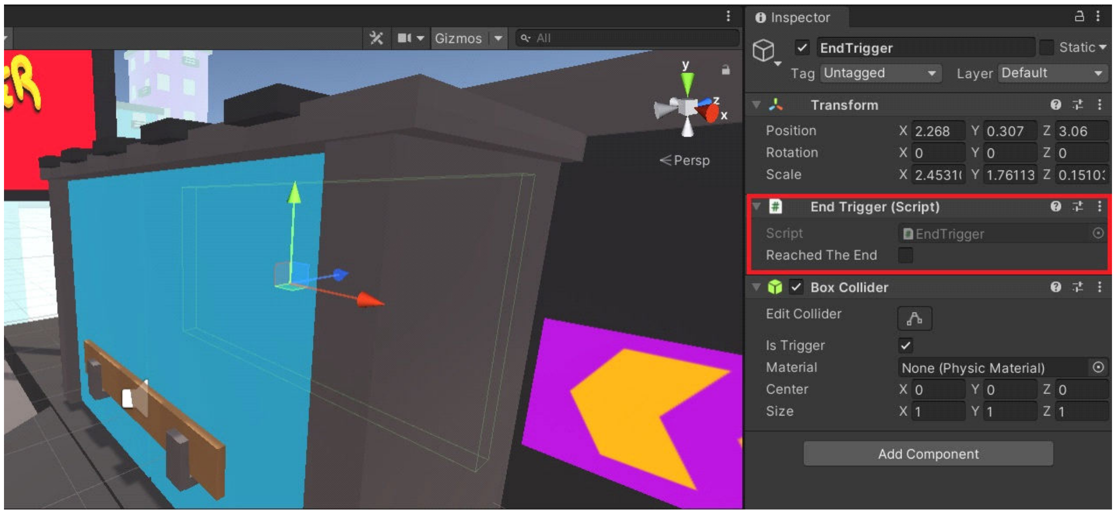
Script	PlayerOutOfBounds
Player Controller	Player_Controller (Player Cc
Ragdoll Player	Player_Controller
Ragdoll Root	Chicken_Ragdoll
Right Hand Controlle	HandR (Hand Controller)
Left Hand Controller	HandL (Hand Controller)
Reset Point	ResetPoint_Start (Transform
Instant Camera Upda	✓

Object Out Of Bound (Script)

Script	ObjectOutOfBound
Obj Reset Point	ResetPoint_Start (Transform
Right Hand Controlle	HandR (Hand Controller)
Left Hand Controller	HandL (Hand Controller)

The end zone uses the same approach.

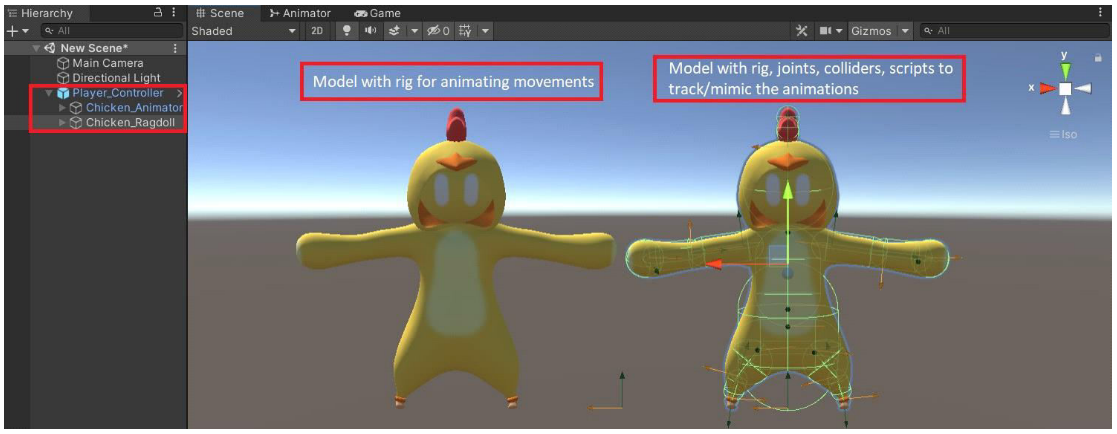
When the Player_Controller enters the trigger we check for the layer "Player" and if so we set some values, we then check those values after the Player_Controller falls through the last fall off zone, if we passed the end zone check and fall off we execute a screen fade, load the menu scene after that.



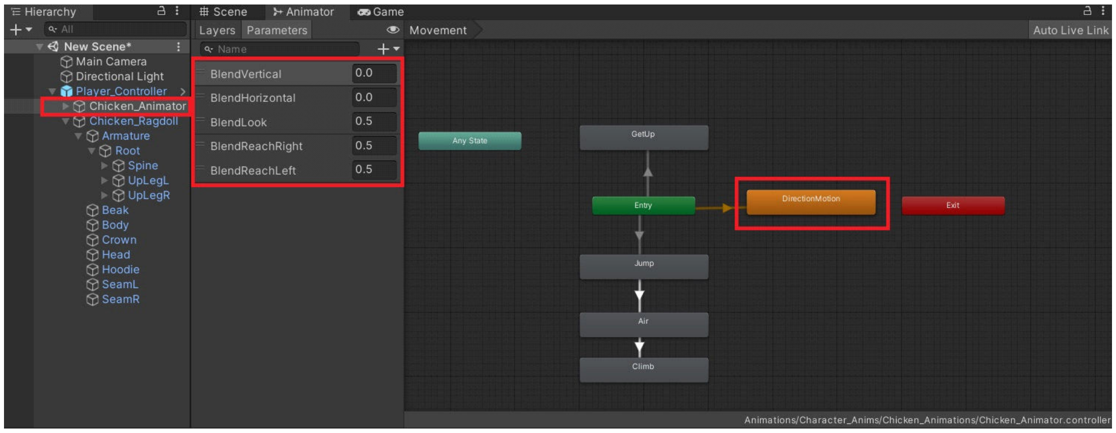
```
Inspector
✓ EndTrigger                         Static
Tag Untagged              Layer Default
Transform
Position     X 2.268     Y 0.307     Z 3.06
Rotation     X 0         Y 0         Z 0
Scale        X 2.4531    Y 1.76113   Z 0.1510
End Trigger (Script)
Script            EndTrigger
Reached The End   ☐
✓ Box Collider
Edit Collider
Is Trigger   ✓
Material     None (Physic Material)
Center       X 0   Y 0   Z 0
Size         X 1   Y 1   Z 1
Add Component
```

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.



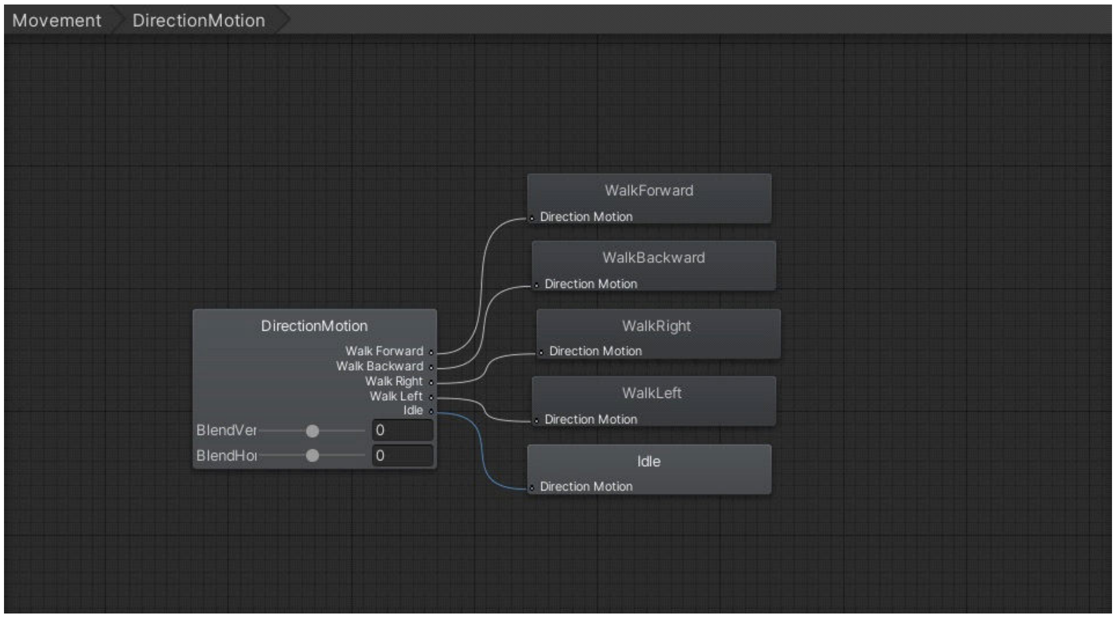
Our animated model uses blend trees, layers and parameters to blend between animations such as reach with the arms and bending of the spine, basically overriding some parts of the animation.



BlendVertical	0.0
BlendHorizontal	0.0
BlendLook	0.5
BlendReachRight	0.5
BlendReachLeft	0.5

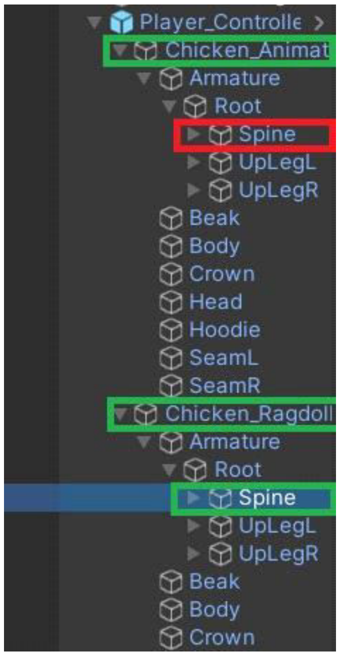
Animations/Character_Anims/Chicken_Animations/Chicken_Animator.controller

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

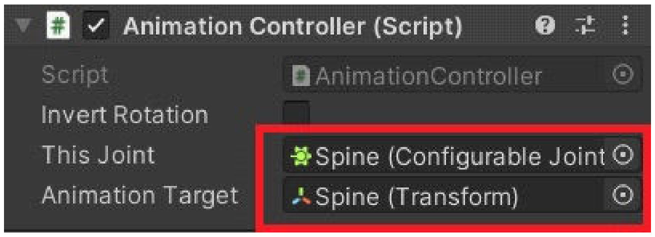


BlendVer 0
BlendHor 0

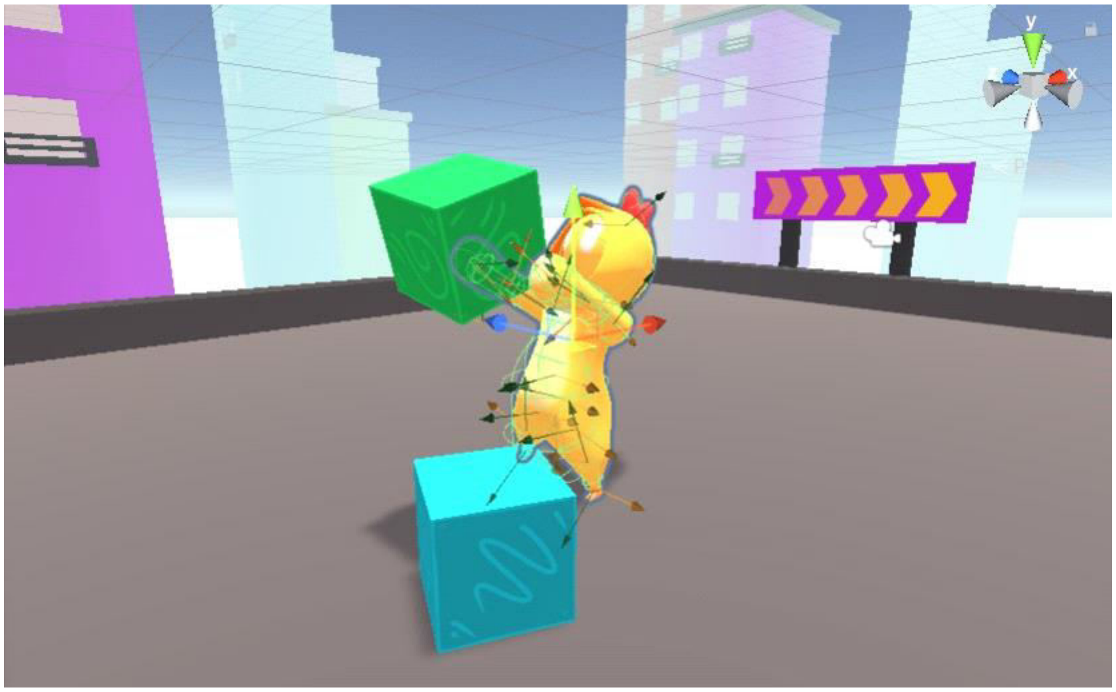
This is how our 2 models work together, we have the animated model and the ragdoll model that have a script on each individual bone that reference which transform to keep track of which is the same bone on the animated model.



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.


Animation Controller (Script)

Script: AnimationController
Invert Rotation:
This Joint: Spine (Configurable Joint
Animation Target: Spine (Transform)



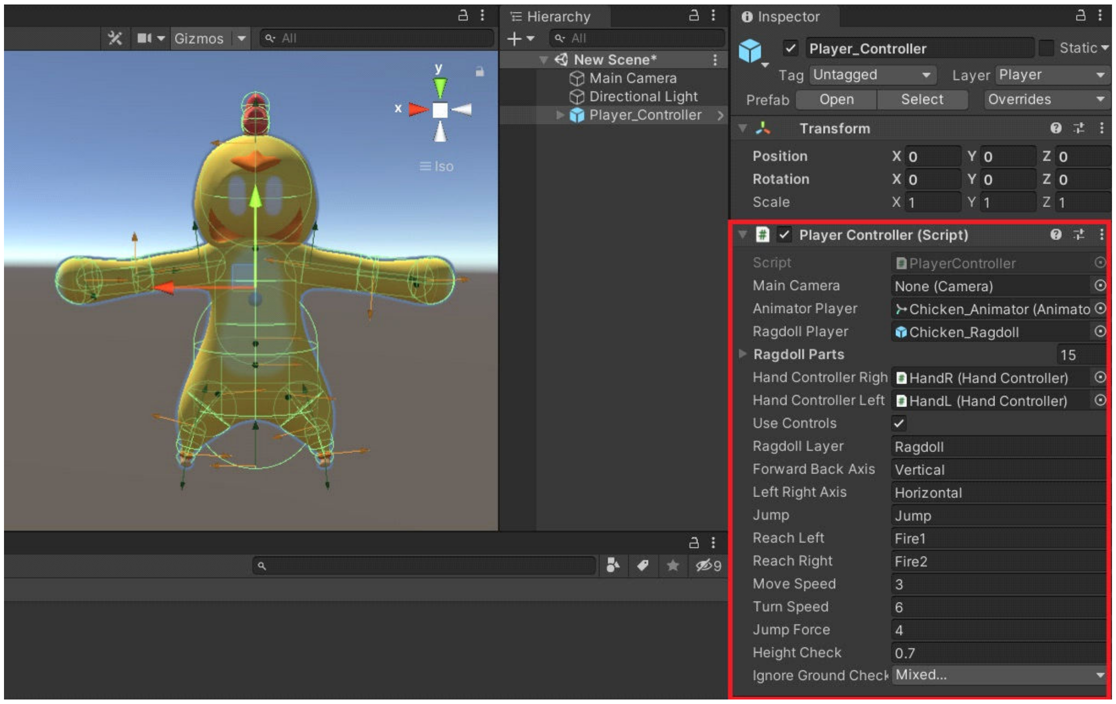
Lets talk about the active ragdoll itself, besides the animation tracking we have some important aspects to cover.

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.



Player_Controller

Tag Untagged Layer Player

Prefab Open Select Overrides

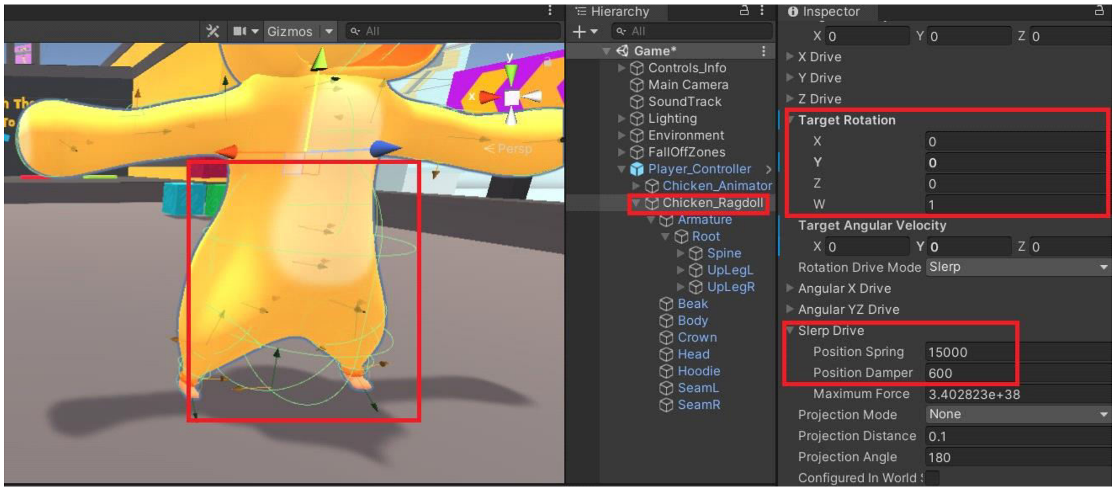
Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Player Controller (Script)

Script	PlayerController
Main Camera	None (Camera)
Animator Player	Chicken_Animator (Animato
Ragdoll Player	Chicken_Ragdoll
Ragdoll Parts	15
Hand Controller Righ	HandR (Hand Controller)
Hand Controller Left	HandL (Hand Controller)
Use Controls	✓
Ragdoll Layer	Ragdoll
Forward Back Axis	Vertical
Left Right Axis	Horizontal
Jump	Jump
Reach Left	Fire1
Reach Right	Fire2
Move Speed	3
Turn Speed	6
Jump Force	4
Height Check	0.7
Ignore Ground Check	Mixed...

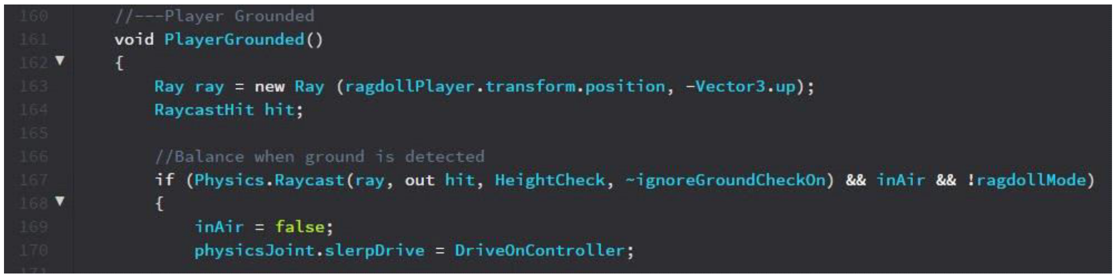
This is where we attached our capsule, using A configurable joint we set the drive high, using the target rotation to keep it from falling over and then also useful for rotating the player.



```
X 0    Y 0    Z 0
X Drive
Y Drive
Z Drive
Target Rotation
    X                     0
    Y                     0
    Z                     0
    W                     1
Target Angular Velocity
    X 0    Y 0    Z 0
Rotation Drive Mode       Slerp
Angular X Drive
Angular YZ Drive
Slerp Drive
    Position Spring       15000
    Position Damper       600
    Maximum Force         3.402823e+38
Projection Mode           None
Projection Distance       0.1
Projection Angle          180
Configured In World S
```

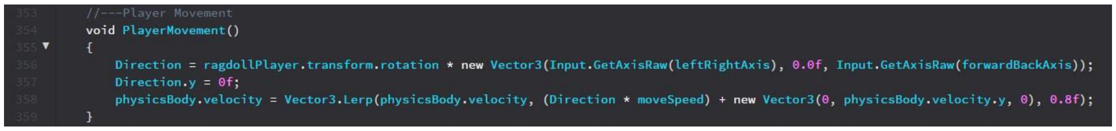
(PlayerController script)

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



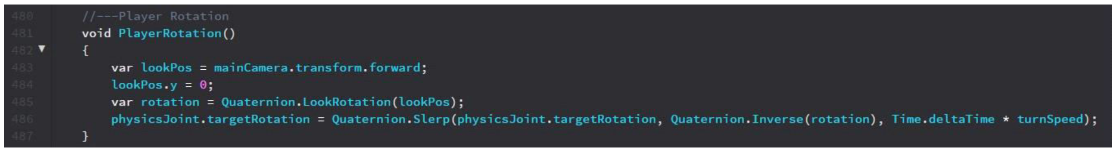
```
    //---Player Grounded
    void PlayerGrounded()
    {
        Ray ray = new Ray (ragdollPlayer.transform.position, -Vector3.up);
        RaycastHit hit;

        //Balance when ground is detected
        if (Physics.Raycast(ray, out hit, HeightCheck, ~ignoreGroundCheckOn) && inAir && !ragdollMode)
        {
            inAir = false;
            physicsJoint.slerpDrive = DriveOnController;
```

we move the capsule with velocity



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

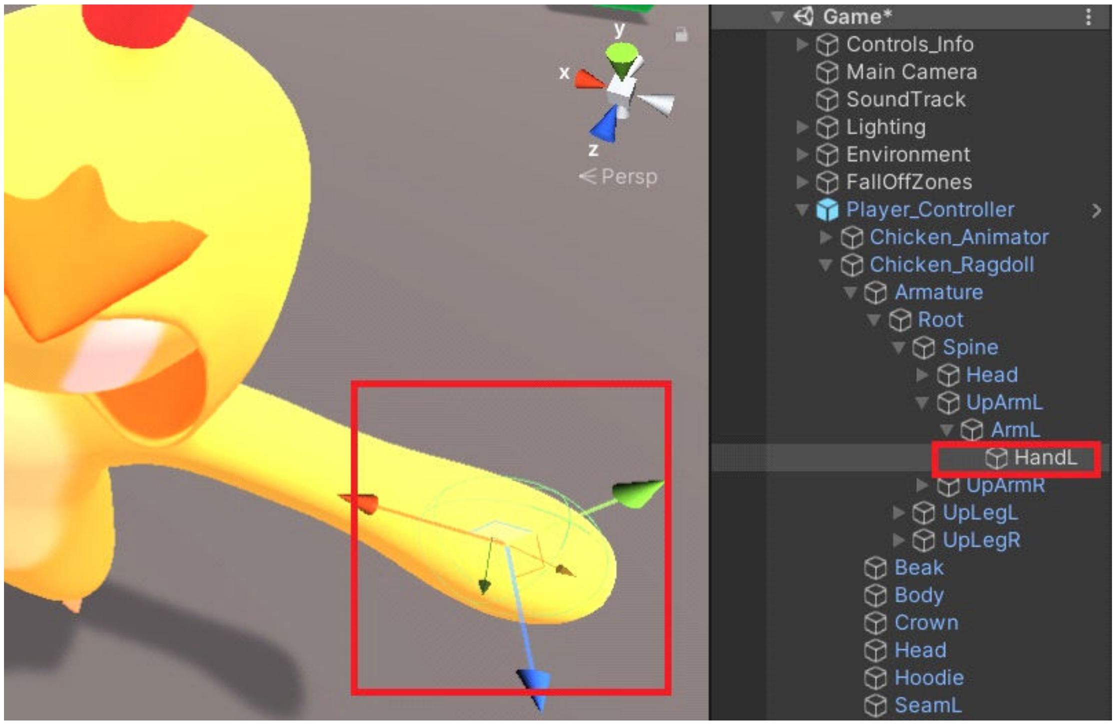
we use the configurable joint's target rotation and set it to our camera direction to rotate our player



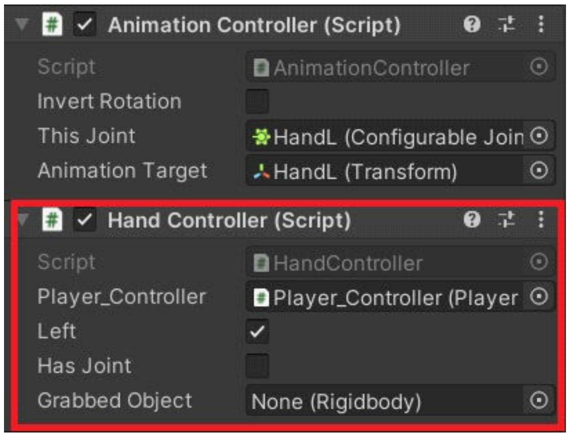
```
    //---Player Rotation
    void PlayerRotation()
    {
        var lookPos = mainCamera.transform.forward;
        lookPos.y = 0;
        var rotation = Quaternion.LookRotation(lookPos);
        physicsJoint.targetRotation = Quaternion.Slerp(physicsJoint.targetRotation, Quaternion.Inverse(rotation), Time.deltaTime * turnSpeed);
    }
```

For the ragdoll hands we have A script for handeling 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

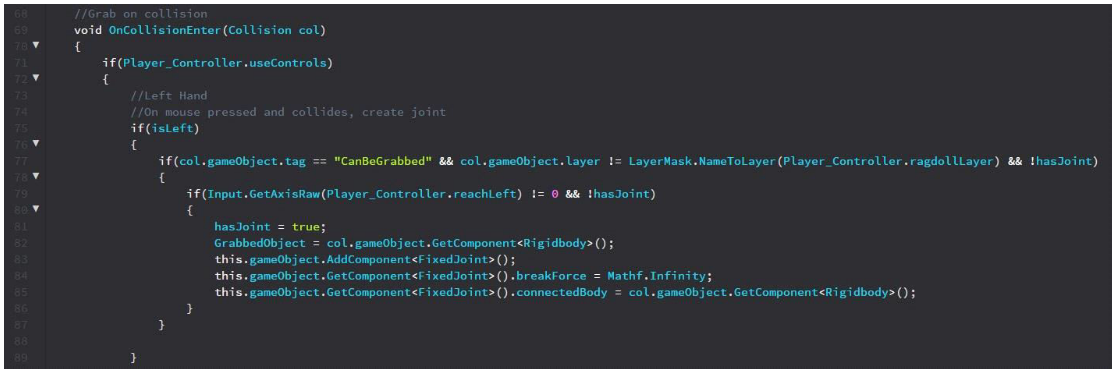


Script	AnimationController
Invert Rotation	
This Joint	HandL (Configurable Join
Animation Target	HandL (Transform)

Hand Controller (Script)

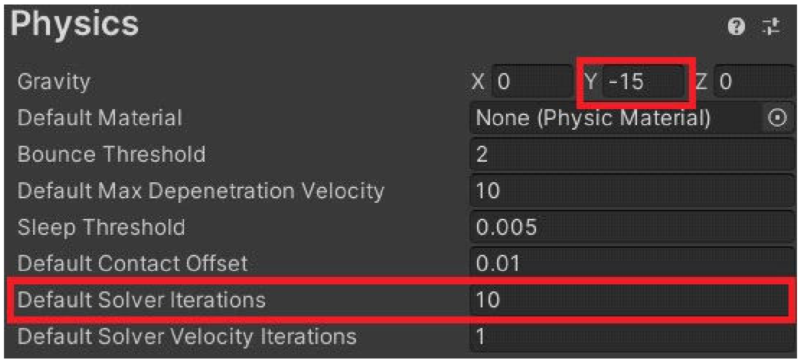
Script	HandController
Player_Controller	Player_Controller (Player
Left	✓
Has Joint	
Grabbed Object	None (Rigidbody)

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.



```
    //Grab on collision
    void OnCollisionEnter(Collision col)
    {
        if(Player_Controller.useControls)
        {
            //Left Hand
            //On mouse pressed and collides, create joint
            if(isLeft)
            {
                if(col.gameObject.tag == "CanBeGrabbed" && col.gameObject.layer != LayerMask.NameToLayer(Player_Controller.ragdollLayer) && !hasJoint)
                {
                    if(Input.GetAxisRaw(Player_Controller.reachLeft) != 0 && !hasJoint)
                    {
                        hasJoint = true;
                        GrabbedObject = col.gameObject.GetComponent<Rigidbody>();
                        this.gameObject.AddComponent<FixedJoint>();
                        this.gameObject.GetComponent<FixedJoint>().breakForce = Mathf.Infinity;
                        this.gameObject.GetComponent<FixedJoint>().connectedBody = col.gameObject.GetComponent<Rigidbody>();
                    }
                }

            }
```

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.



Physics

Gravity	X 0 Y -15 Z 0
Default Material	None (Physic Material)
Bounce Threshold	2
Default Max Depenetration Velocity	10
Sleep Threshold	0.005
Default Contact Offset	0.01
Default Solver Iterations	10
Default Solver Velocity Iterations	1