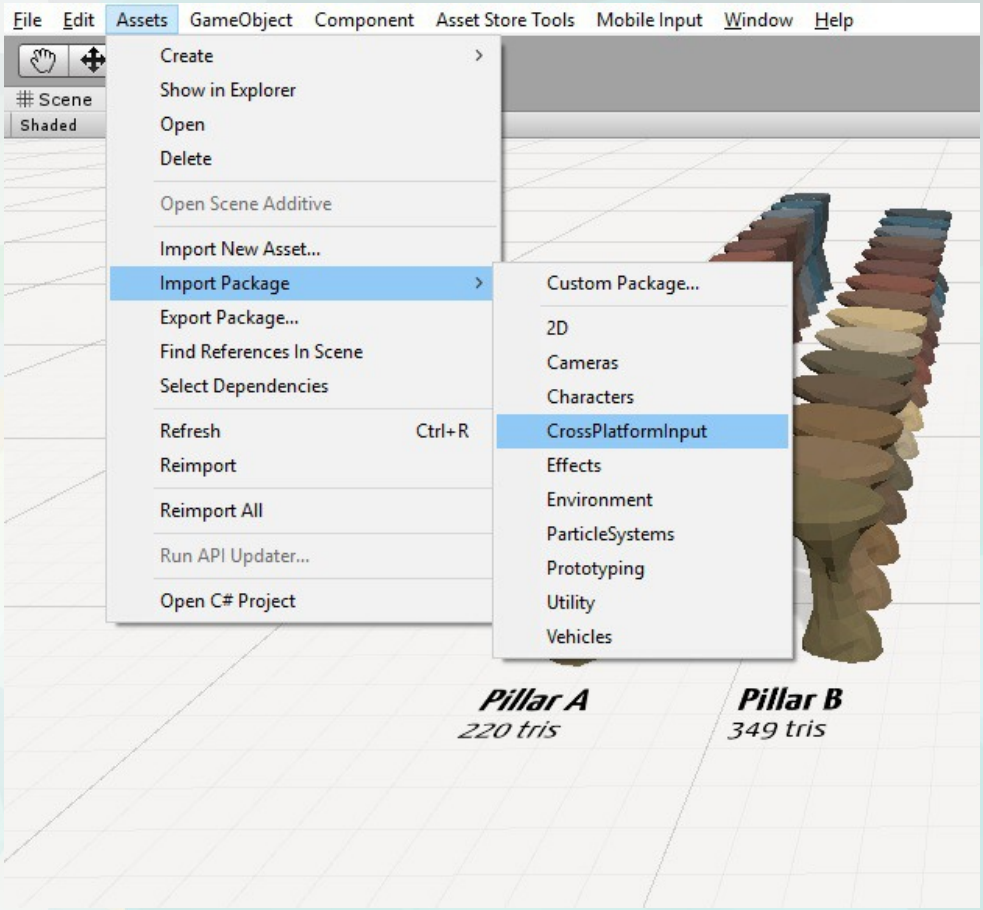
LOW POLY CAVE ENVIRONMENT

by Attila Zold
www.greenworks-productions.ro

REQUIRED ASSETS & PACKAGES:

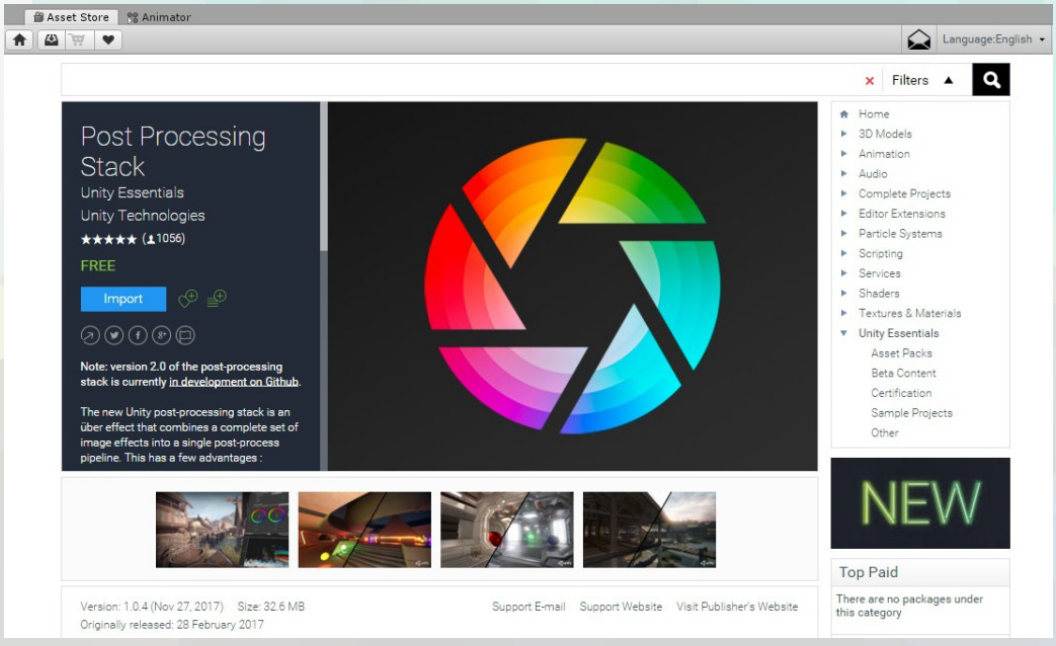
The **PLAYER CONTROLLER** script requires the "**CrossPlatformInput**" package to be imported into your project.



This is a Unity Standard Asset, which means you can import it from inside the Unity engine.

See attached screenshot.

The **POST FX ENABLER** script requires the "**Post Processing Stack**" package to be imported into your project.



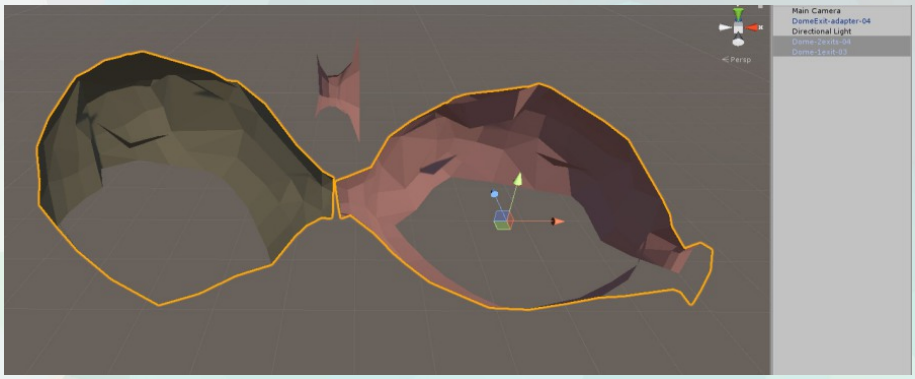
Post Processing effects add extra effects like bloom, depth of field, vignette, ambient occlusion, etc. to your Camera.

This package can be found on the Unity Asset Store.

See attached screenshot.

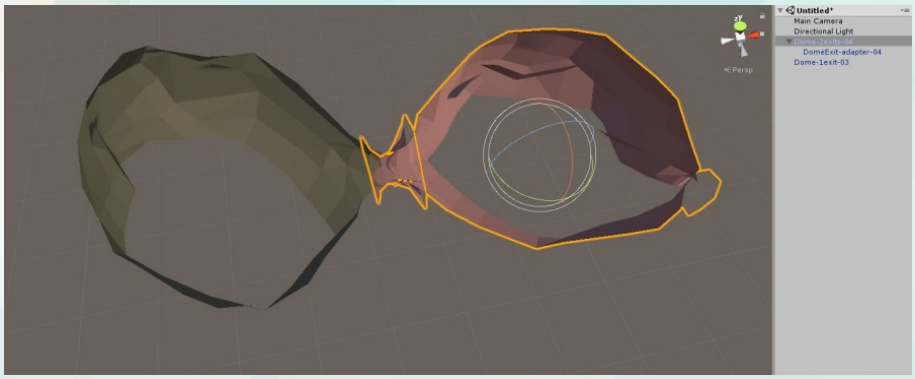
ASSEMBLING CAVES & TUNNELS

ADAPTERS come in handy when your two elements's exits (domes, or tunnels) don't match perfectly.

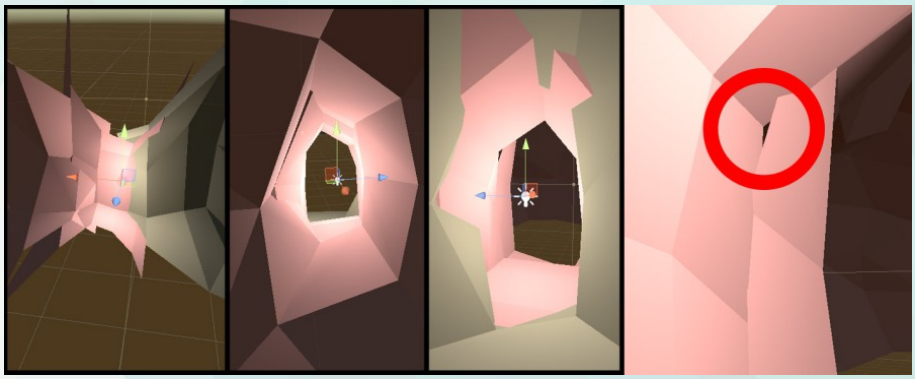


In the **Assets/Prefabs/Domes and Tunnels** folder you will find the building elements for your cave's "walls".

Sometimes when the exits don't match perfectly you will need to use a "**DomeExit-adapter**" prefab, and place it in the middle of the two elements, scaling and rotating it until there's a perfect transition from one element to the other.



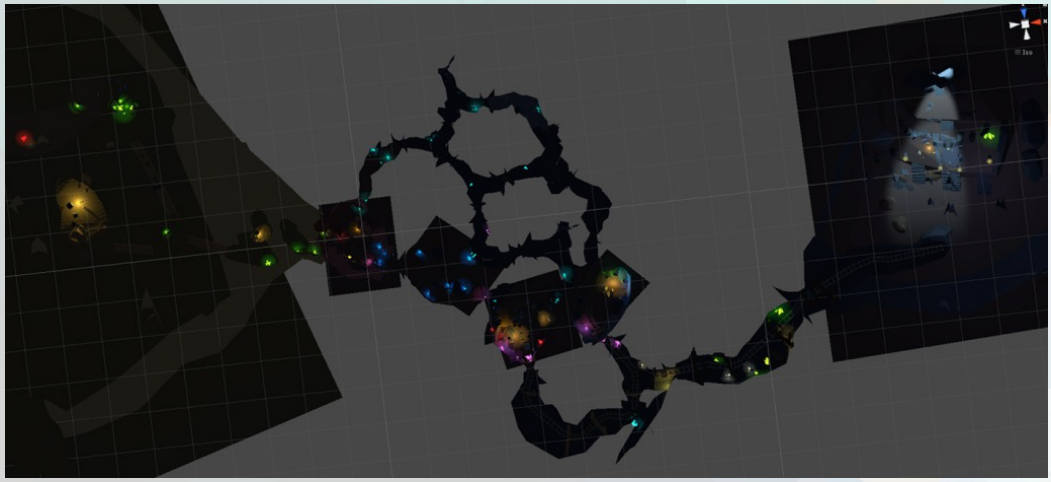
I prefer to parent these Adapters under one of the Domes (or tunnels), because that way when you move the Dome, the adapter follows along nicely, maintaining it's position.



TIP:

Using a "**Point Light**" around the adapter makes it easier to **spot HOLES**, which you need to cover, if you don't want your Player to see through the wall when passing by an exit.

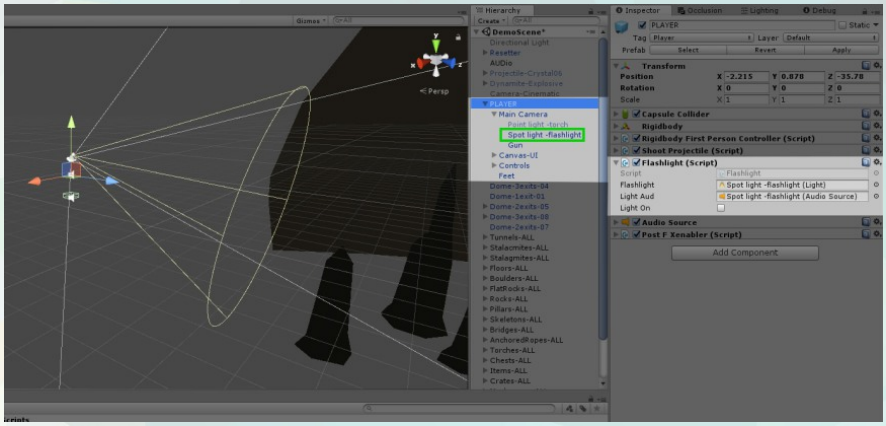
I hope you will combine these elements to create new and exciting levels for us to explore!

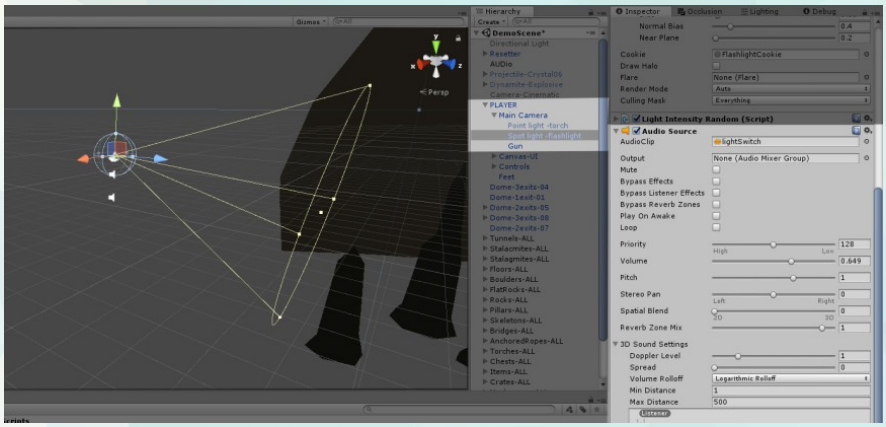


SCRIPTS EXPLAINED

FLASHLIGHT

Turning a Light (preferably Spotlight) on/off with the simple press of a button.





You can place the script on whichever object you desire. For demo purposes, it's on the Player.

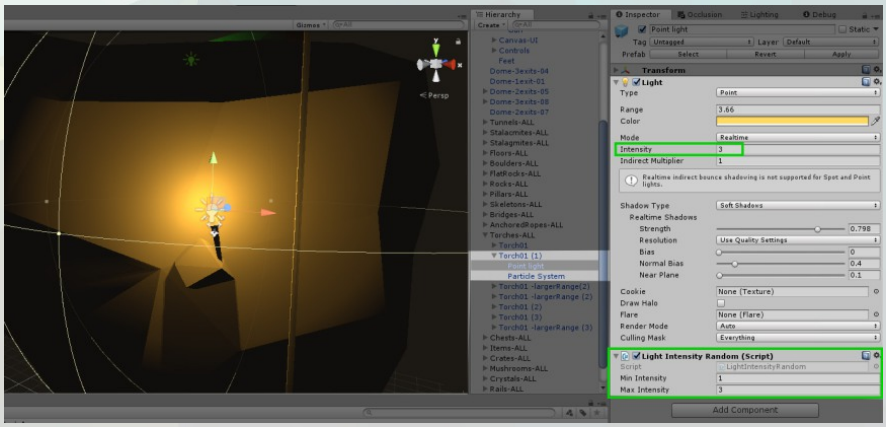
The actual Light should be a Spotlight, to shine light only inside it's cone. This "**Spotlight-flashlight**" is **parented under the Main Camera**, so it will shine in the direction where the camera is facing.

Assign your Light gameobject to the "Flashlight" field.
Assuming your AudioSource (light switch sound) is a component of the "Spotlight-flashlight"gameobject, as in the screenshot below, you can drag the same "Spotlight-flashlight"gameobject into the "Light Aud" field.

"Light On" shows the bool condition of the "Flashlight" being enabled, or disabled.

LIGHT INTENSITY RANDOMIZER

Randomly changing a Light's Intensity value, between the Min and Max values.



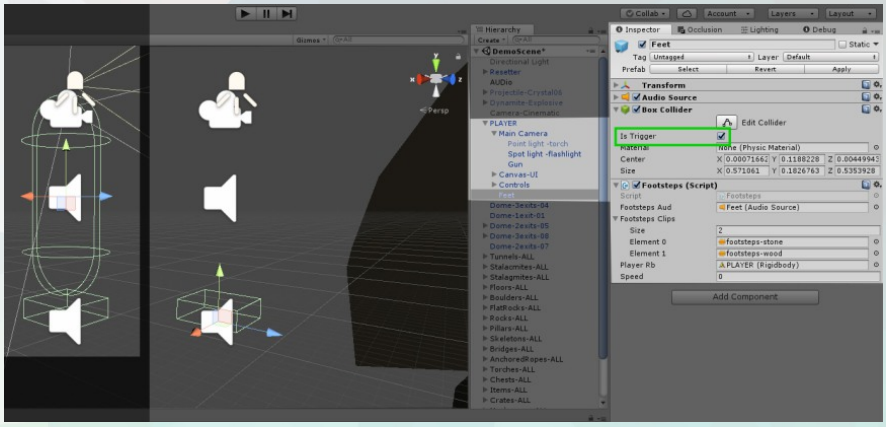
You can place this script on whichever Light gameobject you desire. It will make the Light "pulsate", giving it a realistic touch.

No matter what value is in the "Intensity" field of the Light component, it will be overwritten by the values of "**Min Intensity**" and "**Max Intensity**" from the script.

All you have to do is play around with these two values, see which one creates your desired effect.

FOOTSTEPS SOUNDS

Play a sound when the player walks, play a different sound when he walks on certain gameobjects.

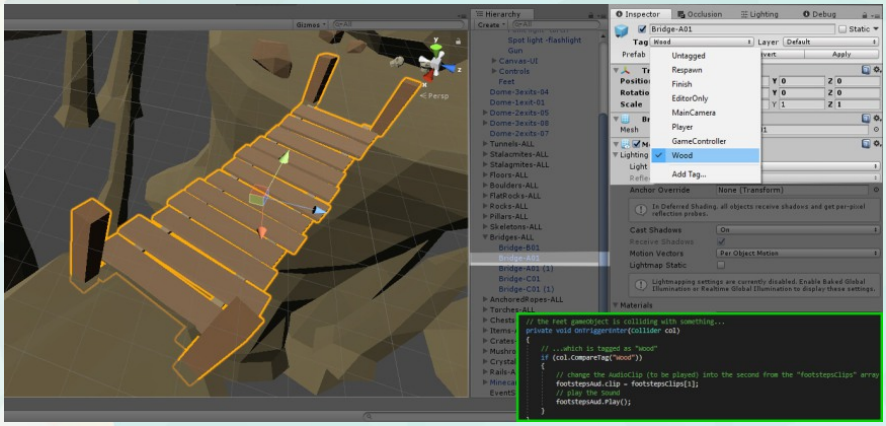


The "Feet" gameobject's Box Collider is outside the player's Capsule Collider, so it will always collide with the ground under.

For this script to work, you will need a gameobject (in this case "**Feet**") **parented under the player**, which has a "**Box Collider**" component that has "**Is Trigger**" **checked**.

You will need an AudioSource (component of "Feet") dragged into the "**Footsept Aud**" field.
You will also need to type at least "1" into the "**Footstept Clips**" **Size** field. If you typed 1, it's the default sound (Element 0: footsteps-stone), and if you type "2" or more, more fields will appear (Element 1, etc) where you can drag additional sounds for wood, or other types of footstep sounds.

Drag your player into the "Player Rb" field (assuming it has a Rigidbody component) , and now the "Speed" value can be calculated from your player's rigidbody's velocity (sound only plays when speed is not 0).



You can add additional sounds types by: 1.adding ex. "Metal" tag to metallic objects, filling out *exaple* at the bottom of script.

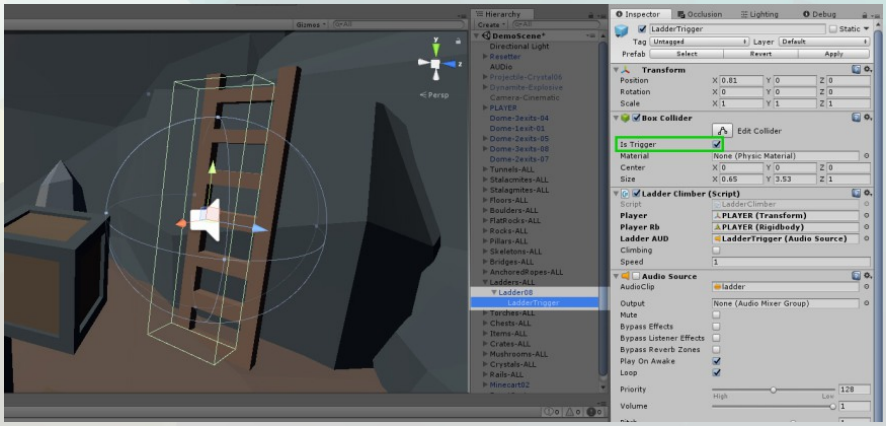
Additional sounds can be setup for various types of walkable surfaces. The second sound is for "Wood", but you can setup any number of sounds.

First of all, you will need to create a tag called "Wood", and assign this tag to wooden object (like the Bridge).

The Footsepts script has an "OnTriggerEnter" function for "(col.CompareTag("**Wood**"))", which plays the "footstepsClips[1]", meaning the second sound clip, "Element 1" field, if you typed "2" in the "Footsteps Clips" Size. Now there's a different sound being played when walking on object tagged "Wood".

LADDER/ROPE CLIMBER

Climb vertically (with "W" or "S") up or down a ladder. Or a rope ("AnchoredRope").



The Trigger gameobject must have a "Box Collider" component, with "Is Trigger" checked.

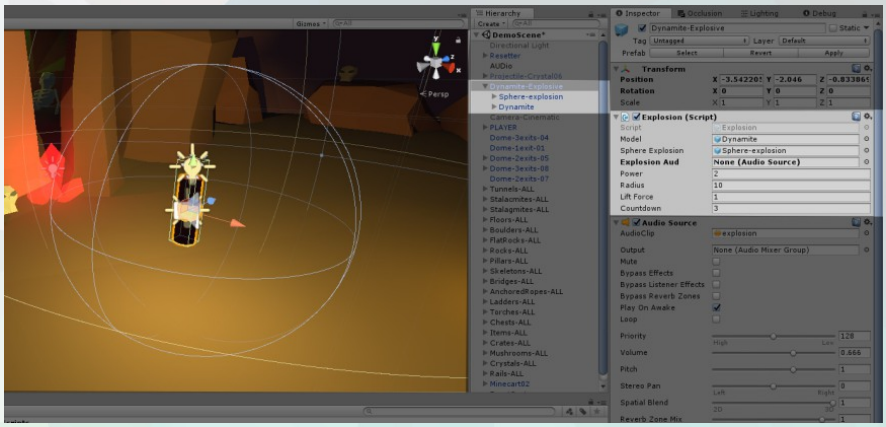
The trigger gameobject, which has a Box Collider (Is Trigger), must also have an **AudioSource** component.

Drag your player gameobject into the "Player" and "Player Rb" fields (assuming it has a Rigidbody component), and the AudioSource into the "Ladder AUD" field. The "**Speed**" value by default is 1, but you can change this if you want your climb to happen faster.

The same script is attached to the "**AnchoredRope**" **Prefabs**, where you should scale the "Rope" and "RopeTrigger" child gameobject accordingly.

DYNAMITE EXPLOSION (PHYSICS)

A gameobject, which detonates after a certain amount of seconds, and affects the surrounding Rigidbodies!



You will find this "Dynamite-Explosive" **Prefab** in the Assets/Prefabs/Items folder.

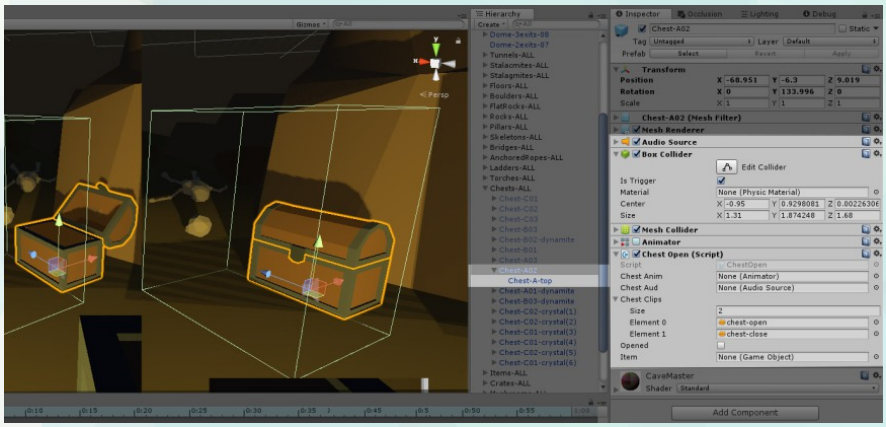
"**Power**" is a value for the **explosion force**, obviously.
"**Radius**" is a value for the **range** in which Rigidbody objects will be affected.
"**Lift Force**" is a value for the dramatic **up-force** you see in movies where objects slightly lift upon impact.
"**Countdown**" is the **number of seconds** passing before detonation.

Happy detonating!

You can also assign this Prefab to the "**ShootProjectile**" script. Because who doesn't like throwing around dynamites?

CHEST OPEN/CLOSE

Walk up to a chest, click, and it opens. Walk away, and it closes automatically.



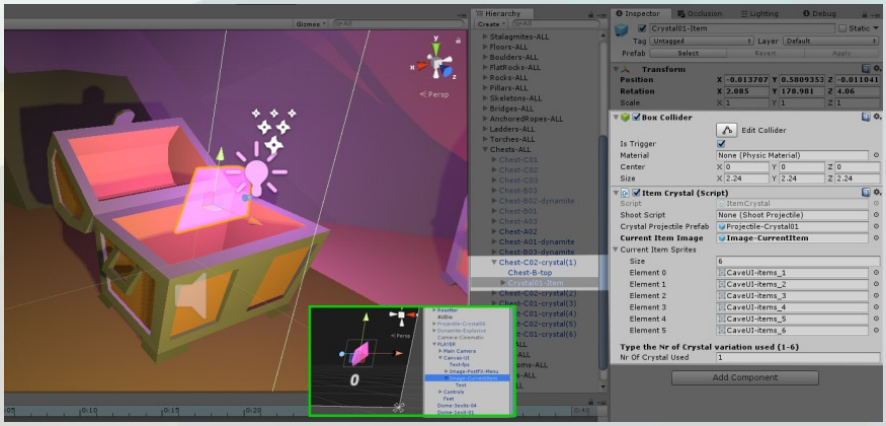
The **Prefabs** from the Assets/Prefabs/Chests folder are already setup accordingly. You can use those.

"Chest Anim" and "Chest Aud" fields will fill automatically upon game start, the "**Item**" field will be addressed in the **section below.**

This script only works if you player gameobject is **tagged** "**Player**". Currently the input for opening is Left Mouse Button ("Input.**GetMouseButtonUp(0)**"), but you can change this inside the script.

ITEM PICKUP/EQUIP

Place items in chest, so the player can grab and use them!



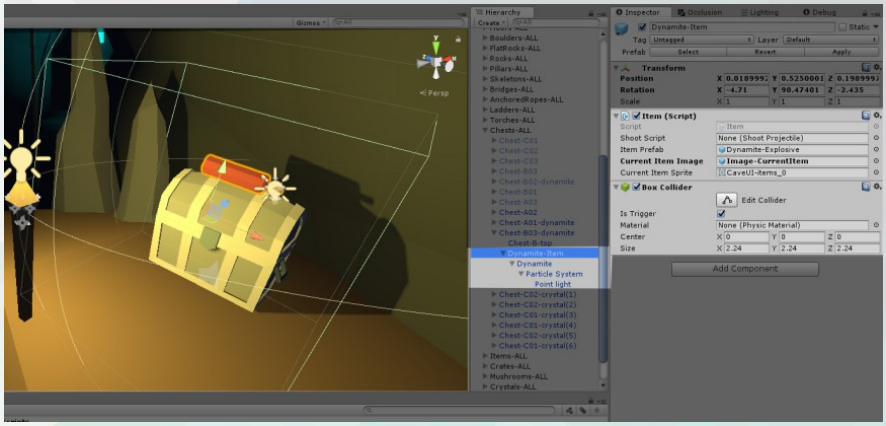
The **Prefabs** from the Assets/Prefabs/Items folder (their names end with ...-Item) are already setup. You can parent them under the Chest, **but they must be ASSIGNED in the "Item" field of the CHEST script.**

"**Current Item Image**" field by default is empty, and you need to assign the UI Image (with UI Text child) here. If you're using the "PLAYER" Prefab, it already has a child gameobject called "**Item-CurrentItem**" (see screenshot). This is an example of what needs to be assigned in the "Current Item Image" field.

Crystal Items have 6 variations. That's why you need to type their number (ex. 1) in the "Nr Or Crystal Used" field, to get the right Sprite from the "Current Item Sprites" array.

WISH TO SET UP YOUR OWN CUSTOM ITEMS? HERE'S HOW:

Items, which use the "Item" script are assigning a Prefab (which you can shoot) to the "ShootProjectile" script.



The Model of this example item is the child called "Dynamite" (see Hierarchy). It's animated, and has a Particle and Light child.

For the sake of this example, let's take the "Dynamite-Item" Prefab, because it's setup

The item gameobject needs to have a "**Box Collider**" with "**Is Trigger**" checked. This is the trigger zone, from inside which the player can grab the item.

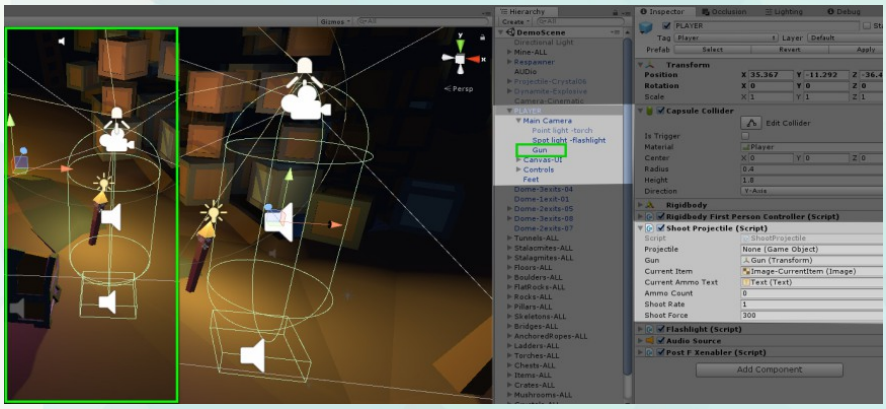
You need to assign a **Prefab** (which we'll discuss in the section below) into the "**Item Prefab**" field. This prefab will be assigned to the "ShootProjectile" script when the item is picked up.

You also need to assign the "**Image CurrentItem**" gameobject (child of the "PLAYER" Prefab) into the "**Current Item Image**" field.

And lastly, please assign a **Sprite** in the "**Current Item Sprite**" field, to be displayed on the UI Image ("Current Item Image").

SHOOT PROJECTILES

This script enables your player to launch projectiles!



Notice the "**Gun**" gameobject, which is **child of the "Main Camera"**, and is placed outside the player's "Capsule Collider".

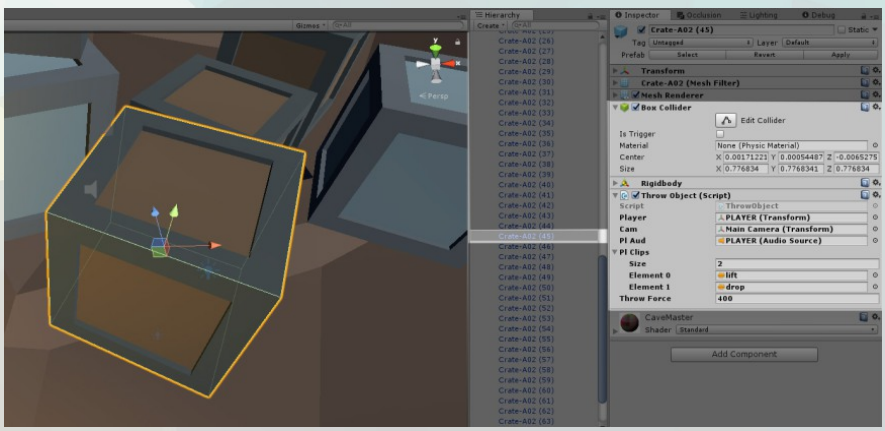
A Projectile Prefab can be assigned in the "Projectile" field (picking up an item should do the same).

The "**Gun**" gameobject, which should be a child of the "Main Camera" (so the instantiated projectile will always follow your camera's position). You will also need to assign the **UI Image** (displaying your current item's sprite) in the "**Current Item**" field, and the **UI Text** in the "**Current Ammo Text**" field.

Ammo count is 0 by default, but this will change automatically when you pick up a shootable item. The "**Shoot Rate**" value is the number of seconds between shots, and "**Shoot Force**" is the force value for launched projectiles (larger values allow you to shoot further).

PICKUP/THROW OBJECTS

Turning a Light (preferably Spotlight) on/off with the simple press of a button.



Only gameobjects with "**Ridigbody**" component work with this script.

This script is placed on gameobjects you wish to carry and throw around. They must have "**Rigidbody**".

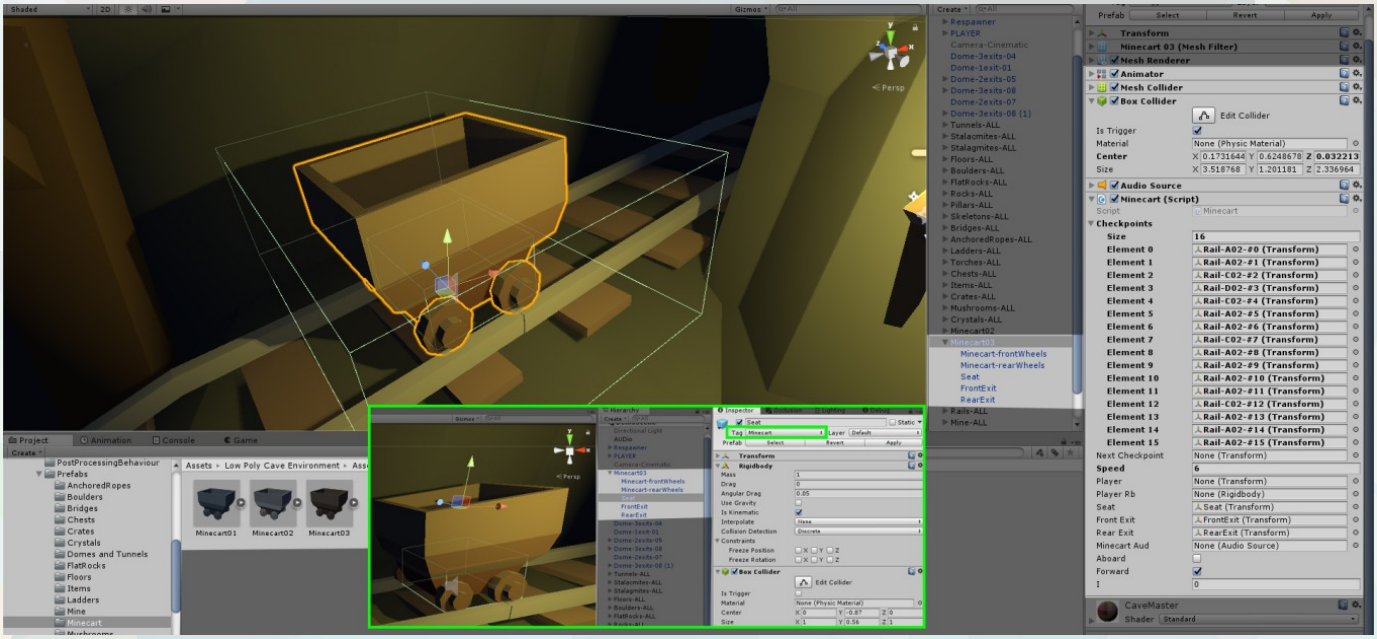
Drag your player gameobject in the "**Player**" field. The "Main Camera" will automatically assign.
Drag the **AudioSource** (preferably on player, because the player makes these sounds) in the "**Pl Aud**" field. Extend the "**Pl Clips**" Size to "2", and assign each sound in the correct field (see screenshot). The first clip will play when the player lifts an object (holding Left MouseButton), the second when he throws it away (releasing Left MouseButton). You can change these inputs from inside the script.

Set your desired value in the "**Throw Force**" field.

MINECART/RAIL PATHFINDING

Build a Rail course and travel on it back and forth with a Minecart.

You will find the MINECART Prefabs in the Assets/Prefabs/Minecarts folder.
They are all setup, like in the screenshot.

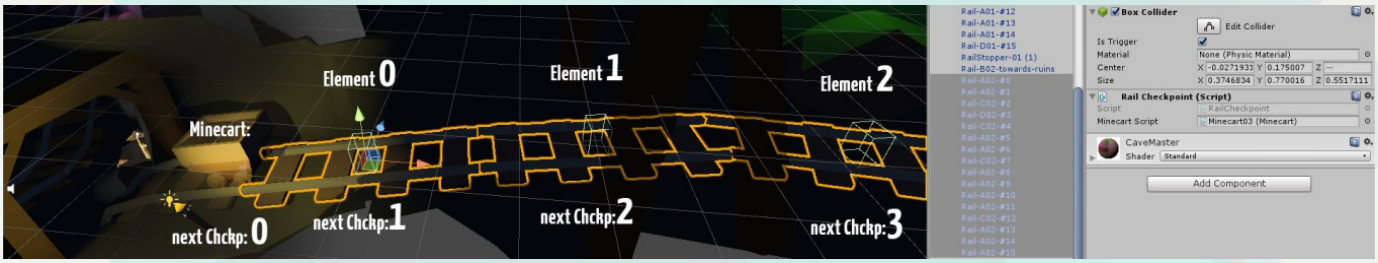


Notice how the "Seat" child gameobject is tagged "Minecart". You will have to create this tag in your Unity 3D, and assign it to your Minecart's "Seat" child gameobject.

The MINECART Prefabs have the following components, required for this script to work:
- **Animator,** which plays the wheels rotation on loop;
- **Box Collider**, marked "Is Trigger", which is the trigger zone, where the Player can climb aboard (LMB is default input);
- **Audio Source**, which contains the "Minecart" audio, and is only played when the Minecart is moving;
- and the **Minecart.cs** script, of course.

Looking at the **Minecart.cs** script, you will notice an array variable, called "**Checkpoints**". This array will store your Rails (those which have "RailCheckpoint" scripts on them). Expand the "**Size**" of this array, by typing in the **number** of Rails you have in your rail course; the fields (titled "Element 0" etc.) will appear automatically and you will need to assign each of your Rails in these fields, in **ascending** order.

The "**Speed**" value is representing the speed at which your Minecart will move. You can change this to any value you like. The "**Player**" and "**Player RB**" fields will assign themselves automatically (assuming your Player gameobject is tagged "Player" and has a Rigidbody component).
The "**Seat**", "**Front Exit**" and "**Rear Exit**" should have the child gameobjects of the Minecart Prefab assigned, and the "**Minecart Aud**" assigns itself too.



The way this script works is pretty easy!

Notice in the above screenshot, how all the selected Rails have the "**Rail Checkpoint**" script attached to them, and in the "**Minecart Script**" field, the "Minecart03" Prefab is assigned (because that's the Minecart from the scene, which will be traveling on the Rails, and will be modified each time it enters the "Box Collider" trigger zone of a Rail).

Yes, the Prefab Rails should have a "**Box Collider**" marked "Is Trigger", just as in the above screenshot, for the "Rail Checkpoint" script to actually work whenever the Minecart (with it's "Seat" child object) enters this trigger zone.

Remember that you needed to assign these Rails in **ascending order** in the Minecart script's "Checkpoint" array fields? If you did that correctly, let's look at what should happen (using the above screenshot as an example).

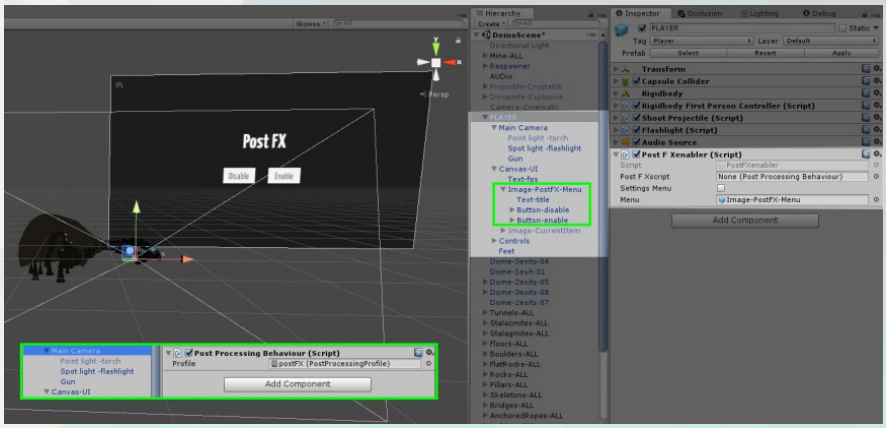
At the start of your game, the "**i**" variable of the Minecart.cs script should be 0 ("**i**" stands for the number of the next checkpoint), meaning the next checkpoint your Minecart will be moving towards is 0, a.k.a. Element 0 from the array, which is your first Rail.

If the Minecart arrives at this Element 0, the "**Seat**"s **collider** enters the **trigger zone** of **Element 0**, (and if the Minecart is moving in "forward" direction) next checkpoint is now +1, a.k.a. Element 1. If the Minecart arrives to this second checkpoint, (still in forward direction) next checkpoint is again changed to +1, now Element 2, and so on...

While **aboard** the Minecart, pressing "**W**" will move you in **forward** direction, thus each checkpoint **adds +1** to your "**i**" number. Pressing "**S**" changes direction, and passing checkpoints **subtracts -1** from the "**next Checkpoint**" array.

POST FX ENABLER

This script brings up a Menu (when Esc pressed) for enabling/disabling PostFX.



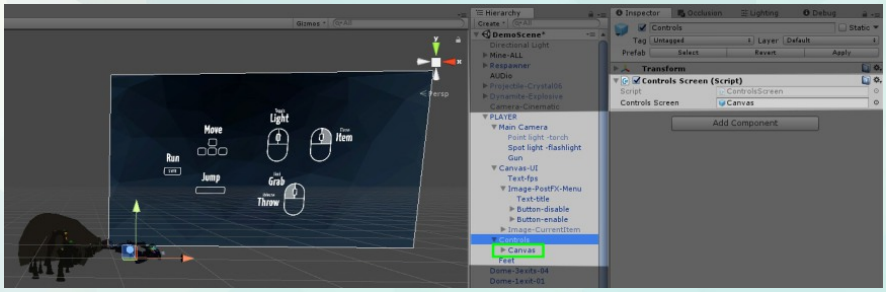
*** This script requires the official "Post Processing Stack" asset from the Unity Asset Store.**

The "PLAYER" Prefab from the Prefabs folder has this Menu all setup, as a child of the "Canvas-UI". You can use that, or setup your own menu.

The "**Post F Xscript**" field automatically assigns itself (assuming you have the "**Post Processing Behaviour**" script on the "Main Camera", like in the screenshot; *the "postFX" PostProcessingBehaviour must be assigned in the "Profile" field of the "Post Processing Behaviour" script*).
You need to assign the UI Menu in the "Menu" field.
If you're using the "**PLAYER**" **Prefab**, this gameobject is already setup, with buttons and all!

CONTROLS SCREEN

Displaying an image of the controls at the start of your game.



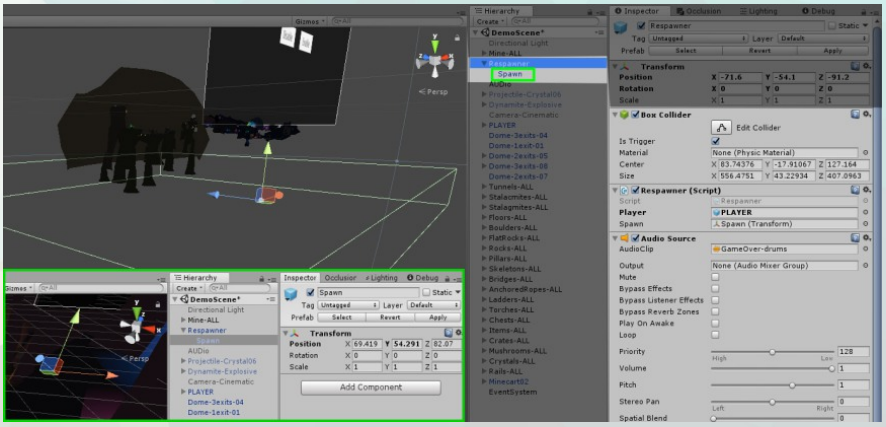
The "**PLAYER**" **Prefab** from the Prefabs folder has the "Controls" gameobject as one of it's children.

Basically it's a fullscreen Canvas with a UI Image. The script starts running at the start of your game, and it **freezes the game time**, while displaying the "Canvas" gameobject.

If the player clicks **Left MouseButton** (input can be changed from inside the script), game time returns to normal and the "Controls" gameobject self-desctructs.

RESPAWNER

Setup a zone where if the player enters, he gets relocated at a Spawn point.



For this script to work, you will need to add a "**Box Collider**" component, with "**Is Trigger**" checked to the gameobject. Edit the Collider's bounds, to fit your desired zone, which if the player enters, he gets relocated to the Spawn point.

An **AudioSource** also must be added, to play a sound when the player gets respawned.

The gameobject, which you will assign into the "**Spawn**" field should be just an empty gameobject **situated at a certain position** in your scene, where you want the Spawn spoint to be.