**Things to Learn:**

**Scope**

**Objectives:**

**Promote to:**

**Research**

**Credits**

# Things to Do

# 0. Intro

*Subtitles are available in English!*

Genius Invokation TCG is a card game in Genshin Impact. It was a fun card game. You start each round rolling 8 dice and use the dice to pay the cost for playing certain cards or actions.

If you want to play Cryo cards, you need Cryo Dice and so on. Very simple and also… very random…

But then, there is this card. **The Jade Chamber**. It makes two of the dice always end up on a particular element. But how do they do this? The dice roll seems to be completely physics-based, so it’s not a prerecorded animation.

In other words: How do you rig the dice? The answer might surprise you.

Welcome to ReForge Labs, where we’ll try to recreate various mechanics from different games to understand how it works.

# 1. Preparation

**Dice Making**

**1.1a**

Let’s start by making the octahedron dice in Blender, also known as D8 for tabletop RPG enthusiasts. This is… surprisingly easy. Just select all vertices and use the **Bevel Tool** in **Edit Mode**.

**1.1b**

Then, we can make the inward faces of the dice using the **Inset Faces Tool**.

**1.1c**

In the **UV Editing**, we used the **Smart UV Project** to quickly unwrap the entire model.

**1.1d**

Then mark every side of the dice to know their orientation. We want all of the arrows to point up.

**1.1e**

And now we overlap all of the UV into a single triangle in the UV Map. This will make it easier to create textures for each of the dice faces in Photoshop, because it makes every face interchangeable. We don’t need to worry about their position in the UV Map.

**1.1f**

Then we separate the materials for each of the faces and the edges. We can then import the textures we made in Photoshop to the faces. Finally, we export this model with the **Path Mode** set as **Copy** so the texture will be included when the model is imported into Unity.

**1.2 - Arena Making**

Now in Unity, we can create a simple plane for the dice to land on and four walls around the plane to keep the dice from falling to the abyss. We can then disable the **MeshRenderer** component on the walls to make it invisible, and align the camera to look down on the plane below.

**1.3 - Dice**

Let’s put the dice model into Unity. The only thing we want to change is the color of the edges. I want it to look like rusty metallic gold, so raise the **Metallic Map** all the way to 1. Awesome!

**1.4 Audio**

For the finishing touch, let’s also create some sound effects for when the dice impacted the ground and when it lit up.

On FreeSound.org, we can find this nice dice roll sound

and chop up the best part in Audacity.

Similarly, the ding sound can also be found here,

but we have to adjust its Bass and Tremble to reduce its pitch and make it more pleasant to hear.

# 2. Regular Roll

With that out of the way, let’s first create the behaviors for a single dice throw.

*Project files can be downloaded for free. Link in the description.*

**2.1**

In a script, we want to randomize the initial rotation, force, and torque applied at the start so every dice throw is unique and different.

**2.2**

Then in the Update() function, we want to check if the dice has stopped moving by checking its rigidbody’s velocity and angular velocity to be zero.

**2.3 -** Find Face Result + Collider Settings

If it has stopped, then we will find which face of the dice is at the top. This is done easily through a collection child object attached to the dice. Using this loop, we can compare all of their Y position values and return the index of the face at the top.

The index is determined in this order, and this order is consistent in this entire project.

**2.4** The headache of Texture swapping order being randomized by Unity

When we find the index, we can use it to swap the face texture to a lit version. We have to use these ridiculous switch-case statements to keep the index order consistent, because Unity randomizes the order of the materials in the MeshRenderer, even though we have set the order earlier in Blender.

**2.5** Sound play + Only once

Anyway, the final part is to play the lit up sound effect with a boolean check so it can only be played once.

**2.5b**

We also add these two functions to play the sound effects when the dice collides with the floor and other dice with the CompareTag function.

**2.6**

Then for a bit of flair, I also added in a Physic Material to the MeshCollider to make the dice more bouncy.

There you go, now we have one complete dice. Using a generator script, we can also Instantiate 8 dice at once. We’re one step closer to the final result.

# 3. Rigged Roll

Okay, so let’s go back to our original question, how do you rigged the dice? The answer, time manipulation. Here’s how it works.

**3.1**

First, we take note of the initial condition of the dice. Then, we roll the dice normally. Record the winning face of the dice, and rewind back to the initial condition. Because we already know the result, we can rotate the dice so the face we want to win is on the original face. Then, as expected, when we roll it, it will end up with the face we want.

This time manipulation can be achieved through the use of **Physics.Simulate** function in Unity, which will do all the physics calculation to an object in a single update frame, so we don’t need to show the player the first dice roll.

**3.2**

Next, rotating the dice to the desired face is a bit tricky. Unlike a regular 6-sided dice, the amount of rotation needed for each face is not the same. But we can create a ScriptableObject which keeps a table on how to rotate to each face.

[Table here, from as row, to as column]

**3.2b**

Now, here’s a cool trick to rotate the dice easily: just separate the MeshRenderer component as a child object! We don’t need to do any complex calculation with Rotations and Quaternions.

**3.3**

And now, we can scale it up to eight dice rolls and create a nice UI to control what we want with the dice. So far so good!

**3.4**

But of course, it’s not that simple, is it? It never is. For 99% of the cases, this method works fine. But it’s not 100% perfect, because Unity’s physics system is not deterministic. That means even with the same initial condition, the final result may vary. The physics has an inherent randomness based on the time between frames and on different devices.

**3.5**

To solve this, we need to make the second dice roll exactly the same as the first one, without any physics interaction. The only way to do that is to record the exact movements of the first dice roll in every frame. That includes the exact time when the dice play sounds.

Then when it is time for the second dice roll, we rotate the dice to end up with the face we wanted and play back the movements of the dice. The result is a perfect animation replayed twice with different results for the dice. That is the dream roll right there! 8 omni dice!

**3.6**

Irrelevant to our goal, but I also create a little fun rewind function. This is easily achieved by playing the animation backward.

Closing

So, there we go! I hope you guys find this video interesting and informative! If you do, then don’t forget to like this video and subscribe for future contents.

If you want to support this channel, I recently opened a Patreon page where you can watch my videos earlier among other benefits. Or you can buy me a coffee on my Ko-Fi page.

And with that I’ll see you guys later, goodbye.