Grant Hoechst
CS 175 Final Project Writeup
12/18/16

# The Physics Rooms

## Controls

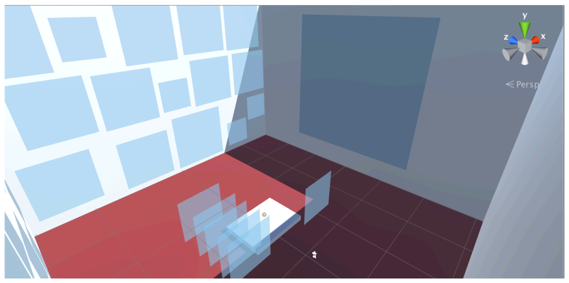
Mouse and Keyboard:
- Move with the arrow keys or WASD keys
- Aim and fire by moving and left-clicking the mouse
- Jump with the space bar
- Sprint by holding the shift key
- Increase the power of your projectile with 'E'
- Decrease the power of your projectile with 'Q'
- Reload the level with 'R'

Xbox Controller:
- Move with the left joystick
- Aim with the right joystick
- Fire with the right bumper
- Jump with the A button
- Sprint by holding the left bumper
- Increase the power of your projectile with the B button
- Decrease the power of your projectile with the X button
- Reload the level with the Y button

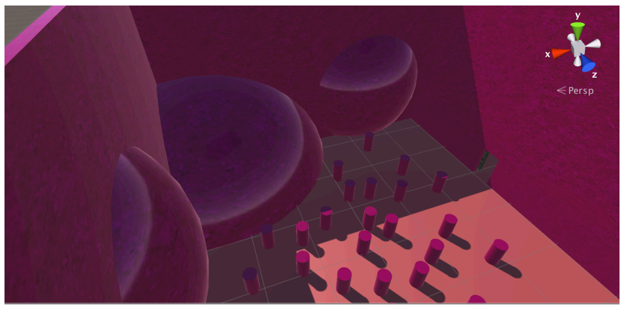
## Project Description

"The Physics Rooms" are a collection of slightly stylized physics-based rooms built in Unity 5. The player is free to walk about three rooms and interact with them by throwing small projectiles, the velocity of which can be controlled. The project is made up of three rooms:

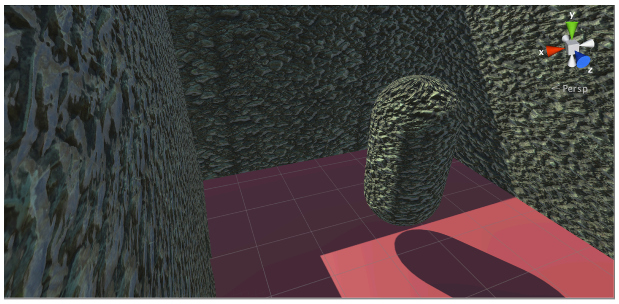
- A Glass Room, with simple destructible glass panes



- A Rubber Room, with a dynamic audio bounce



- A Sticky Room, where projectiles stick to the walls with a satisfying “splat”



The project has two primary focus points: interactivity of the material, and dynamic audio.

**<u>Interactive Materials</u>**

The three materials, along with the projectiles, have several custom scripts written in C# that drive their functionality.

The **Glass** room is made up of several destructible panes. Each pane has a bluish, transparent material, allowing the camera to see through the pane to a certain degree. Upon being hit by a projectile, a pane can “shatter,” breaking into several small pieces and falling to the ground. This shatter process is semi-dynamic: upon collision, the solid pane is destroyed and replaced with a pre-modeled collection of smaller shards, each of which has its own collider and rigidbody, and so is individual affected by gravity. Upon collision, an explosive force is applied to the shards at the point of impact to simulated the shattering. This force is dependent both on adjustable parameters (power, radius, and upward component of the explosive force), and on the relative velocity of the projectile: in other words, the glass will shatter more explosively when hit with a higher-velocity projectile.

Each pane also has a breakability threshold, making stronger panes can be created that do not break unless under high-impact (the large pane on the right of the starting room is one such pane. The glass is also programmed to be breakable by any object, including other glass shards, meaning that a cool cascading effect can be created by shooting a pane higher up on the wall. Upon collision, an Audio Source is also created at the point of contact for 3D audio (discussed more in-depth later).

All of this functionality is written by me from scratch in the “Break.cs” file, and the “FallingGlass.cs” file simply destroys the shards on contact with the floor, to ensure a more efficient run.

The **Rubber** room is coated in a pink, textured, rubbery material, which is much more elastic than the material of the other rooms. Projectiles will bounce off the material, and will bounce more the more velocity they have going into the bounce. This pink material is not perfectly elastic (although it could be made so), meaning that projectiles lose some velocity on each bounce. I included lots of curvy surfaces to create interesting bounce patterns.

The main allure of the rubber room is its dynamic bounce audio. Along with being fully 3D audio (discussed later), the pitch of the bounce sound created by each impact is proportional to the impact velocity of the projectile. In a very stylized, cartoony way, higher-velocity impacts will result in higher bounce sounds. The sound clip itself was created by me, and the pitch of the sound is adjusted in “Boing.cs.”

Finally, the **Sticky** room has a green, textured, bump-mapped material that makes each projectile that touches it stick to it. This can be used to draw fun pictures, or simply to enjoy the splat sounds of the room. This functionality comes from "Stick.cs," wherein any projectile that comes into contact with the material is converted to a kinematic object, meaning it is unaffected by any forces, including gravity or prior momentum. In this sense, each object is frozen in space upon collision with the material, creating a satisfying halt upon contact. Again, this room is also programmed with 3D audio.

Other scripts written by me include a script to reload the level, a script to fire projectiles of varying velocity (with sound effect if desired), and the scripts needed to control the game with an Xbox 360 controller instead of a mouse and keyboard.

## **3D Dynamic Audio**

The other big goal of my project was to implement satisfying 3D audio for each room, as well as a dynamic score that shifted depending on which room the player is in.

Each collision, whether with shattering glass, rubber bounce, or sticky goop generates a 3D audio source to play a sound clip. In the case of the rubber, that clip is a single clip that is dynamically repitched; in the case of the glass and rubber rooms, a clip is chosen randomly from a series of preselected clips. By generating a 3D audio source, the collision sounds have position in space relative to the player. This position determines both *falloff* (collisions from further away sound quieter and vice-versa), and *pan* (collisions from the left sound from the left and vice versa in stereo). To test this, try shattering a pane of glass close up and far away, or try throwing a projectile and rapidly turning so that the impact happens to the side. This functionality is best experienced with headphones.

Along with the collision audio there is some persistent audio that is centered with the player, and so always sounds the same volume. There are simple footsteps and jump sounds when walking around. There is also a persistent musical score (composed by me) that changes to reflect the feel of each room upon entrance. This is controlled by "MusicControl.cs," a script I wrote to recognize large triggers placed in the environment and to control Unity's AudioMixer so as to switch between room sounds. A 'stinger,' or short attention-grabbing sound, is also randomly chosen and played upon entering each room.

## **What I Learned**

This project was an intriguing way to engage with the concepts we saw in 175 at a slightly higher abstraction level. Having an understanding of materials, bump mapping, animation, and scene graphs/ hierarchy helped me dig in immediately to the Unity framework, whereas it would have been a little more obtuse otherwise. It also made me curious for the possibilities beyond what I've worked on here, and if Unity is a system that would allow for a lower level of abstraction or not.

More than anything, I came away with a new knowledge and appreciation for Unity as a higher-level way of organizing objects in 3D space. I also came away with what I think is a very important skill: the ability to sift through an API documentation to garner the information I need to achieve a certain task. That's not really something I've needed to do before, but is something I get the sense will be hugely helpful in a number of areas, both within and outside the field of Computer Graphics.

If I were to continue working on this, I would love to add:

- Glass that reappears on its own after a time
- Functionality so that the balls disappear on their own after a time
- More kinds of rooms!

**Sources**

With the exception of the glass shattering and the sticky splats, which were downloaded from copyright-free websites, every sound and music clip was created by me.

My primary source was the Unity Scripting Documentation. I searched heavily through the description of the scripting API for information on the various methods I needed to write my scripts, as well as for descriptions of the AudioMixer to get my soundtrack mixed properly.

I watched a YouTube tutorial to help with the glass breaking functionality:
https://youtu.be/dA-MPeZ0HVM

The First-Person Character Controller I used is a member of the Unity Standard Assets. However, as I said, I programmed the Xbox 360 Controller functionality.