Miles Kang
Christopher Kha
Joshua Young

Introduction

Our project can be viewed as two separate components: a record player simulation and a shooting game. It starts off as a seemingly simple record player that can play music, but upon clicking the ‘smash’ button, destroys the current disk and transitions the scene into a more complex shooter minigame. We used a Git workflow to collaborate efficiently on the project and tried our best to maintain good coding habits to avoid merge conflicts and unwanted side effects. Challenges along the way were resolved by getting together and discussing bugs and potential solutions.

Record Player Simulator

The record player simulator consists of a record player stand, a vinyl record disk, and a rotatable needle. The user can interact with the simulator through the buttons on the control panel. The basic controls provided by these buttons are needle unlocking, needle rotating, disk spinning and volume adjusting. If the disk is spinning and the needle is locked while it is on the disk, one of two songs will begin to play depending on the position of the needle. If the disk is stopped or the needle is unlocked, the song will stop playing. The volume from the simulator can be adjusted with the (-) and (+) buttons and adjusting the volume will also change the position of the volume slider on the record player stand.

Designing the record player simulator involved an in-depth knowledge of basic objects, transformations, JavaScript audio functions, and methods for interacting with the scene using key triggered buttons. For example, the ‘spin disk’ button both pushes down the physical button on the record player and starts rotating the disk, with a subtle wobbling effect accomplished using a very small random rotation along the z-axis. The ‘(un)lock needle’ button animates the needle object along its vertical axis and also makes a check for the disk spinning and the rotation position to determine whether music should play, and which song should play. In a similar fashion, the ‘rotate left’ and ‘rotate right’ buttons check the current angle of the needle and move to the next position, or stay put if the needle is already at the far left or far right or if the needle is in a locked position. A difficulty here was keeping the needle position from slightly shifting off from our desired coordinates after a large number of interactions. With JavaScript’s float-oriented math operations, it is not always safe to check for equality, since the value we are checking could have a value that is off by a tiny fraction from the expected result. We resolved

this by checking for a range of values and then hard-assigning the desired value if the condition is met.

One challenge that we faced when using the Javascript audio class was debugging an issue where the second song would not play if the first song had previously been played and stopped. We tested many different sections of our code in order to find the error and discovered that the issue was due to an interaction with Chrome where an audio object would continue loading the audio file even when it is paused unless the file path is cleared after pausing.

Finally, we are able to transition to shooting game mode by clicking on the ‘smash’ button. This triggers an animation sequence of shattering the disk, translating the needle to the center of the record player to function as a tank cannon, and dropping a new disk onto the record player to start playing upbeat game music. This entire sequence consisted of a shattering effect, which is explained in further detail in the ‘Target Shooting Game’ section, and some fairly straightforward translations.

Target Shooting Game

The goal of the target shooting game is to hit targets (music note blocks) and score points within 30 seconds. Shooting a target awards 12 points while firing a projectile costs 2 points so hitting a target awards a net gain of 10 points. The player has a maximum of 3 shots which recharge over time. The game ends when time runs out, a projectile hits the player, or if 100 points have been accumulated. We will now detail the game objects and mechanics used to implement our game.

The first mechanic used in our game is collision detection. We implement collision detection using axis-aligned bounding boxes. This means that each object with collisions enabled had two points, a minimum and a maximum point, which defined a cuboid around it. This cuboid was then checked against the cuboid of every collision-enabled object to see if they intersected at the x, y, and z axes. We used an object oriented design for our objects which allowed each object to perform specific actions upon colliding with another object while having all objects inherit basic collision detection.

Next, we used particles to simulate shattering objects. When a projectile collides with a target, we generate particles by spawning projectile fragments and target fragments with pseudo-random velocity vectors and rotation angles. These particles are affected by gravity and disappear upon hitting the ground.

The final mechanic is object physics. We implement two classes ‘Body’ and ‘Simulation’ which help generate realistic physics in our game. The ‘Simulation’ class decouples the frame rate and the physics simulation in the scene. This means that if the frame rate jumps higher or lower, physical forces such as gravity are still applied at a constant rate and keep the object’s movement smooth. We once again use object oriented design for our objects to determine whether an object is affected by gravity or not.

The game objects used are the tank, projectiles, walls, and targets. We will first talk about the tank object and its movement. The tank consists of three major subcomponents: the record player, disk, and needle. We moved the tank around the scene by treating these subcomponents as children for the parent tank object and only applying transformations to the parent. The ‘w’/ ‘a’/ ’s’/ ‘d’ controls move the tank left, right, up and down using translations along the x and z axes. The ‘q’/ ’e’ controls rotate its cannon left and right by applying rotations along the y axis of the child needle object. We also used a sinusoidal function to apply small translations along the z axis of the tank for a hovering effect.

Next, we have the projectiles shot by the tank. The projectiles are created by scaling down the disk on the record player. An initial direction vector for the projectile is determined using the direction the needle is pointing in. The ‘space’ key fires a projectile, and the ‘t’ key toggles gravity on and off for future projectiles.

The walls are game objects which create a boundary for the scene. There are six walls including the ceiling and ground. When projectiles collide with any of the walls, they properly ricochet by calculating the angle with respect to the wall’s normal and adjusting their velocity and rotations correctly.

The final game objects are the targets shot by the player. There are up to twenty targets at a time on the screen. When a projectile collides with a target, a realistic shattering effect is created using particles as detailed above. Targets respawn every five seconds in pseudo-random locations. We determine where a target spawns by selecting a random coordinate on the x-y plane in front of the player. If a target has already been created on that spot, we ignore it and choose a new location. Otherwise, a target is spawned with a random value for the z-axis to get targets of varying depth.

One challenge we faced while implementing this game was properly bouncing projectiles off walls. If disks were shot at certain angles, they would slip through the walls and exit the viewable scene. After testing the collisions for several hours, we were able to narrow down the issue to not rotating the collision boxes of the projectiles correctly. Another challenge which we were not able to overcome was adding shadow mapping to our project. We wanted to implement

a standard two-pass algorithm which would render our scene from the viewpoint of the light to create a depth map and then render the scene again using the depth map to test for shadows. Unfortunately, the tiny-graphics library abstraction of WebGL made it difficult to properly draw our scene twice without completely rewriting the library functions.

Git Workflow

To collaborate, we created a remote repository on Github and cloned local branches of it on our machines. We worked on separate parts of our code to minimize conflicts as much as possible, but we occasionally had to resolve them manually by looking thoroughly at the code and making sure that certain features did not break others. This was usually not a huge issue because we were pretty good about frequently pulling each others' changes and letting each other know when major commits were pushed.

When we compressed our finished project, we found that the zip file was 170 MB while CCLE has a file size limit of 100 MB. We eventually discovered that the .git folder made up 100 MB of those 170 MB. We resolved this by creating a copy of the project folder, deleting its .git folder, and compressing it.

Textures/Sounds

All of the music, sound effects, and textures in our project are completely original. The textures were drawn using Autodesk SketchBook, and the music and sounds were produced using FL Studio 20.

Member Contributions

Joshua Young

- ➢ Implemented collision detection, object physics, and particles.
- ➢ Implemented game mechanics for firing projectiles, spawning targets, and shattering targets/projectiles.
- ➢ Implemented game time, points system, and first/third person camera switching.

Christopher Kha

- ➢ Created all textures.
- ➢ Created disk and needle objects.
- ➢ Implemented customization on disk object based on shapes of revolution to generate fragments.
- ➢ Debugged issues with Javascript audio.

Miles Kang

- ➢ Implemented record player functions (needle locking/unlocking, needle rotation, volume controls, etc.)
- ➢ Created the record player object as an obj file using Maya.
- ➢ Implemented the transition animation sequence, with the falling disk and the needle moving/scaling.
- ➢ Implemented dynamic key triggered button controls in both the record simulation and game.
- ➢ Composed the music and sound effects.