Nilava Saha
09/22/2023

Exploring a Game Engine

Introduction

We were tasked to create a 3D room filled with balls. Each of these balls will have 100% elastic collision so they should bounce off each other and the wall. There should be at least 3 balls in the space participating in the collisions. The room must have 6 textured walls to box the balls and the balls themselves should be textured as well. I used the default Godot sprite that the engine provides you to texture the balls and the wall. Although it may sound like it would make the map confusing the textures overlapping each other helps the user understand the edges of the room and how textures work on balls allowing us to differentiate between them. Aside from that directional light on the balls, as well as a point light creates visual shadows that allow us to differentiate between the balls and the wall even better.

Coding and Additions

The coding for this project was quite challenging because of the lack of documentation and chatGPT was not able to guide us through this was quite lacking. But after listing out the requirements like:

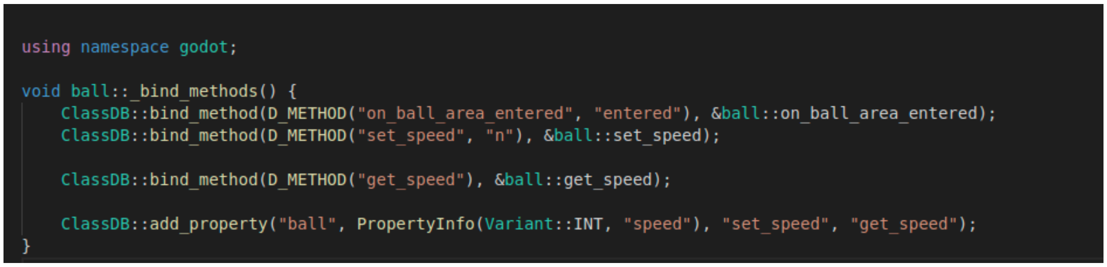
Extension Ball

- Speed
- Direction
- Collision
- Spawn Locations
- Setter Methods
- Getter Methods

Extension Wall

- Normal Force
- Setter Methods
- Getter Methods

After listing and understanding each portion of the extension I was able to get going. Starting with the ball. Creating the speed variable is easy I just needed a double that held the speed variable. But simply having the speed variable wasn't enough. I found myself needing to modify speed quite a bit as I was coding and creating the objects in Godot. So in order to expedite the process without needing to recompile just to update the speed I added a way to add it in as a class property so it was easier to modify as I was messing around in the editor.

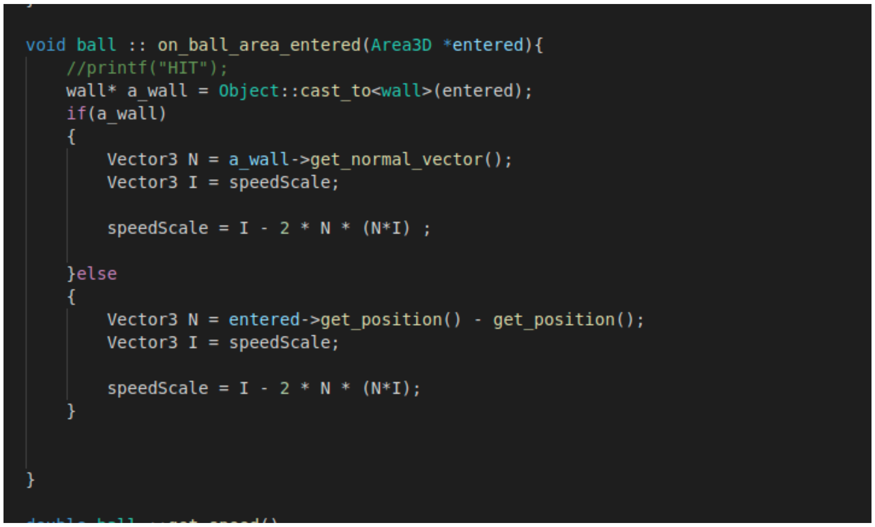


```
using namespace godot;

void ball::_bind_methods() {
    ClassDB::bind_method(D_METHOD("on_ball_area_entered", "entered"), &ball::on_ball_area_entered);
    ClassDB::bind_method(D_METHOD("set_speed", "n"), &ball::set_speed);

    ClassDB::bind_method(D_METHOD("get_speed"), &ball::get_speed);

    ClassDB::add_property("ball", PropertyInfo(Variant::INT, "speed"), "set_speed", "get_speed");
}
```

Nilava Saha
09/22/2023

Next, direction. In order to figure out what way the balls would move. I created a vector3d that generated a random number between -5 to 5. This would determine which direction and how fast compared to the normal speed the ball would be going. This created the random speed and the random direction of the ball's movement. And with spawning we did the same thing. The balls had the ability to spawn within 300 units of the x, y, and z direction from the origin. And all you had to do to add more balls was drag in another ball scene into the editor.

I am fairly proud of my collision methods because it's so simple and bare. All the calculations do is use the normals from the ball and the wall to calculate the direction the ball should be going during collisions. The only issue was figuring out how to parse the incoming Area3D into a possible wall object. And since I know that there are only walls and balls in this space there is no need to check for other objects.



```
void ball :: on_ball_area_entered(Area3D *entered){
    //printf("HIT");
    wall* a_wall = Object::cast_to<wall>(entered);
    if(a_wall)
    {
        Vector3 N = a_wall->get_normal_vector();
        Vector3 I = speedScale;

        speedScale = I - 2 * N * (N*I) ;

    }else
    {
        Vector3 N = entered->get_position() - get_position();
        Vector3 I = speedScale;

        speedScale = I - 2 * N * (N*I);
    }


}

double ball :: get_speed()
```

With wall, I just needed to save Vector3 in order to do that I added an X, Y, and Z property to the Godot editor so I could change it in the app. This was needed to set the normal force of each wall. I created a getter method to turn these separate variables into a singular vortex3 variable.

<u>Camera Movement</u>

The camera should spawn at the origin point. It also has the ability to move around by:

- W - move forward
- S - move back
- A - strafe left
- D - strafe right
- Arrow keys - rotate direction
- Q - move up
- E - move down

Nilava Saha
09/22/2023

Understanding Code Layout

```
/*
This portion of the code would include all the imports and is needed for the code to function.
IE its .h or .hpp files.
*/
Class
{
/*
This portion of the codes will usually contain the methods bindings required for it to functions
*/
/*
This portion of the codes will contain the main methods such as _ready _process or class calls
*/
/*
This last 3rd part of the code will include any additional methods I created for this program to
run. ie. getter and setter methods, calculations, etc
*/
}
```

Issues

The biggest issue was trying to figure out each method needed for something to work. Connecting when area_entered was an issue because it depended on me binding methods that I created. As well as the syntax required for it to work was a hassle to figure out. Additionally adding the property into Godot was another hassle because of syntax once again. All of this took me 2-3 hours of documentation scrolling and videos watching just to figure out that I should also be calling additional methods inside of the calls. Just understanding syntax and getting that to work was a complete hassle.

Additionally in the beginning I wanted to set it up on my personal computer because I compute to campus. But after trying for 2 days I gave up and moved to the CS Linux computers cause I didn't want to figure out pip and scons installing on a Windows machine.

Running the Program

The rundown is provided in the readme. But as a TLDR paste the file into your project folder. Scan the project folder using Godot. Run the project called demo and pressing the play button, not the scene play button, should allow it to run. There is a video included for you to watch if it doesn't work.

Conclusion

A 2-week project requiring us to learn how to modify the godot engine using GDExtension and CPP by learning each of the methods required to get it working.