Brew Balls Post Mortem

Building Pong

One morning, I decided to build Pong to review manipulating rigidbodies in Unity. In the process, I learned a little bit about Physics Materials and their relationship with rigidbodies.

Because I used Physics Materials onto the Ball game object, I did not have to calculate a new vector with every bounce. The ball was simply given an inherent bounciness that was handled conveniently through unity's physics system for me. Nice.

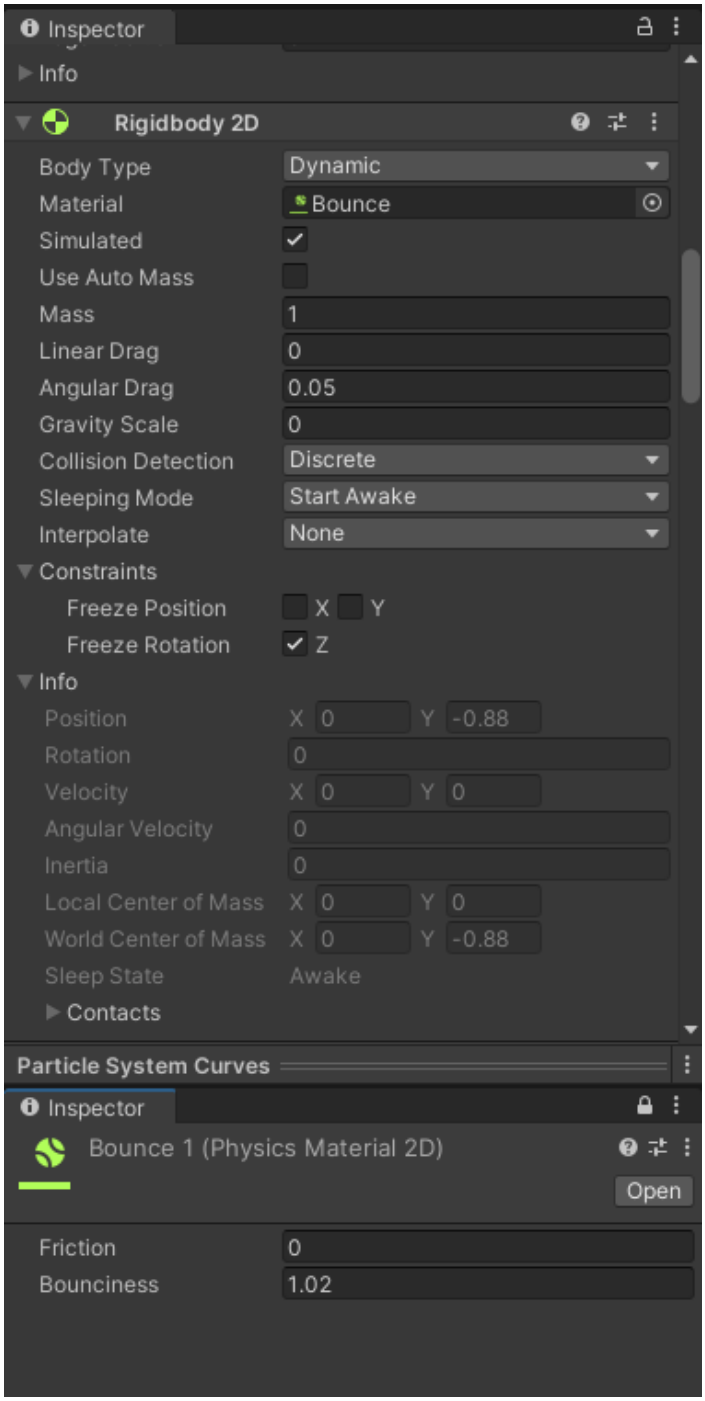
After that, I created a function for the ball to launch from.

Launch() simply chooses if it wants to throw the ball to the left or right, then it decides if it wants to throw the ball up or down. These combinations of forces will send the ball into one of four diagonal directions.

```
2 references
private void Launch()
{
    float x = Random.Range(0, 2) == 0 ? -1 : 1;
    float y = Random.Range(0, 2) == 0 ? -1 : 1;
    rb.velocity = new Vector2(speed * x, speed * y);
}
```

This function is called when the game first starts, and again after a goal when the ball respawns.

Top this off with an OnTriggerEnter2D for each player goal to move the ball back to the center and add to and print an increase to the player's score, and you have a simple Pong recreation.



However, I began to get the itch to expand on this…

Making Pong Better

After finishing my simple pong clone, I started to wonder about what I could do to improve pong. When I asked myself ***"What is the worst part about pong?"*** I immediately thought ***"Waiting for the ball to come back to you".***

In order to improve the constant downtime of pong, I tried to think of ways I could **give the player a new decision to make** while the ball is leaving your paddle and eventually making its way back to you.

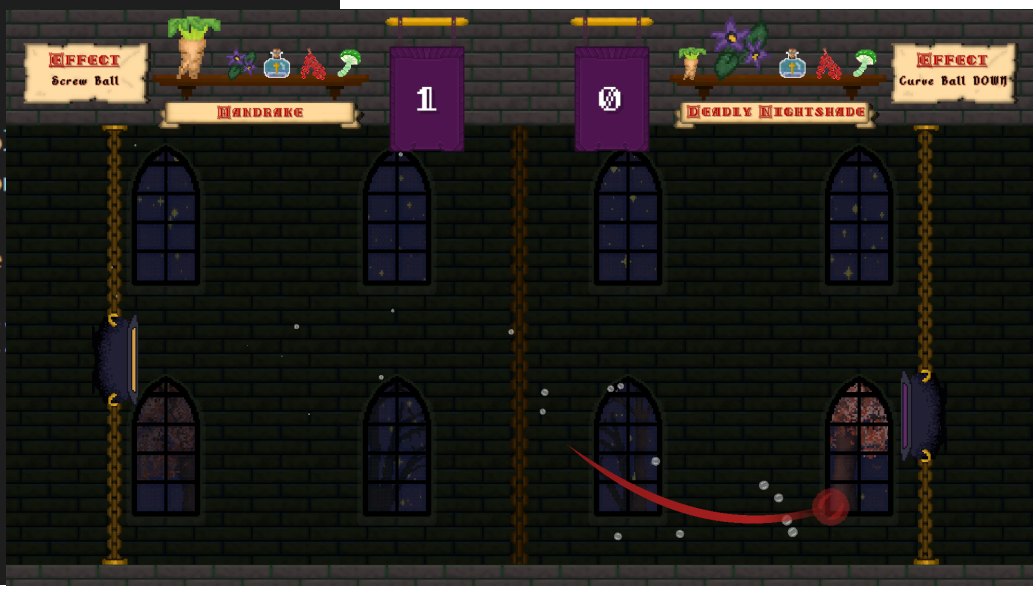
This was a few days after my school's game design club (SGDA) had kicked off their latest game jam with the theme Pick Your Poison. This started to lead my thinking on this "between-turn decision" to be about picking between a series of selections. And that is how I got the idea to add pseudo-baseball style pitch patterns that the player can add to the ball on the fly to throw off their opponent. The easiest of these was the curveball.

```
2 references
void CurveBallDown()
{
    ballShellSprite.color = colorCurveDown;
    trail.colorGradient = colorCurveDownGradient;
    //SHAKE CAMERA
    StartCoroutine(cameraShake.Shake(.1f, .1f));
    //HIT STOP
    FindObjectOfType<HitStop>().Stop(0.1f);
    //Play VO
    vo.CurvePicker();

    rb.gravityScale = .75f;
}
```

```
2 references
void CurveBallUp()
{
    ballShellSprite.color = co
    trail.colorGradient = colo
    //SHAKE CAMERA
    StartCoroutine(cameraShake
    //HIT STOP
    FindObjectOfType<HitStop>(
    //Play VO
    vo.CurvePicker();

    rb.gravityScale = -.75f;
}
```

When the player chooses the curveball (either upwards or downwards) it sets the Ball's rigidbody to a value, or the negation of that value, depending on if it is meant to curve upwards or downwards.

Once I realized I was onto something, I began to research more into baseball pitch styles. And while none of them were exactly exciting enough themselves to put into the game, the names of the pitches gave me some inspiration.



Pick Your Poison

Since the “Poison” selection was using horizontal inputs to contrast with the paddles’ vertical controls, I thought it would be easiest for the player to feel oriented if the “Default” ball style was in the middle, and you could scroll left or right to swap to the styles. So for the sake of symmetry, I decided to go with 5 poisons so that the default could be in the middle with 2 selections to make on each side.

Screw ball
It’s a more intense curve ball that chooses its direction AND intensity randomly.
rb.gravityScale = Random.Range(4f, 6f);
rb.gravityScale = Random.Range(-4f, -6f);

Curve ball down
rb.gravityScale = .75f;

Default ball (normal pong)
rb.gravityScale = 0f;

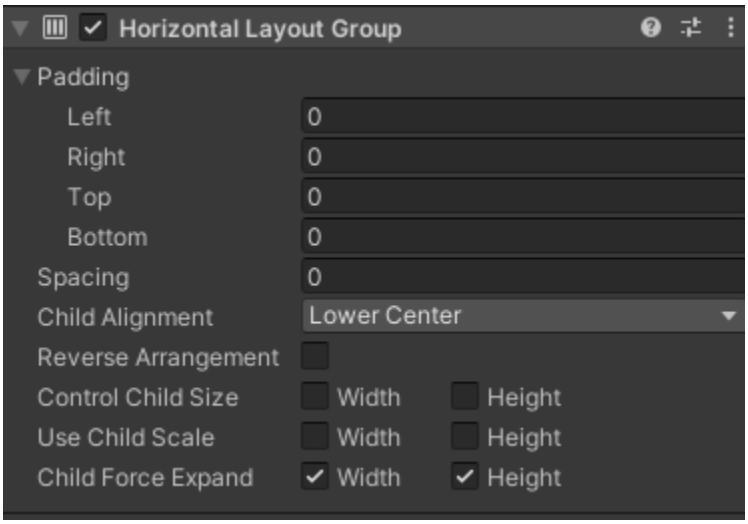
Curve ball up
rb.gravityScale = -.75f;

Fast ball
rb.gravityScale = 0f;
rb.velocity = new Vector2(20, screwAngle);
The X of the new vector is positive or negative depending on if it is used by the left or right player.

After every hit, the player’s poison selection goes back to the middle (**default ball**). This forces the player to consider different effects every “turn” rather than just staying on one. Cause that would defeat the whole purpose of making a new decision every hit.

Final(?) Thoughts

While the game jam is over, I do not think I am quite done with this project. I have received massive amounts of helpful feedback and I would like to use that to make some changes before porting this game over to the UTD Maker Space Arcade Cabinet. I have wanted to put a game onto an indie arcade machine for a long time now, and I developed Brew Balls with the arcade experience in mind. Everything from the control layout to the soundtrack.



I learned a lot about Unity's canvas system. Such as the ability to use Horizontal Layout Groups to keep even spacing between UI elements. ***(Thank you Mikey Bess for showing this to me)***

This system is how the potion ingredients stay centered on the player shelf when the selected one enlarges.

Next Steps

This is usually the part of the Post-Mortem where I would talk about what I would have done differently, but for a game jam project, I really have no regrets.

The screen shake is a little much, the main menu soundtrack is too similar to the main theme, and the screw ball is imbalanced (but still fun).

The addition of a Player VS. CPU mode was a bit of a rushed afterthought, and has an odd jittery bug right now, but I am sure it's because of the odd way I chose to implement.

I will look into cleaning up the code for this as well as creating difficulty modes using the "***AIBrain***" float values I created to determine the chance of the AI making moves.