Design Document

Oliver Leisinger
260644
WS 2021/22
Prima

Some_sort_of_Pinball_thing

Table of Contents

Idea

This project is supposed to be a pinball type game that takes advantage of being digital by including different pickups and power ups. Just like a real pinball machine, the player can interact with the game using the spring that launches the ball and the flippers to prevent the ball from getting removed.
Taking advantage of digital objects being able to phase through each other, the flippers can rotate 30° in both directions.

Arena

The arena is made out of the case, consisting of barriers as well as ground and glass to keep the balls inside of it. Inside there are 5 bumpers which give points and push the ball away, if it collides with them, 3 pyramids that are just in the way to diversify the way the balls move as well as coins, which give a lot of extra points when all are collected within a short time and different power ups. Then there are also the 2 flippers that the user can rotate using WASD and the arrow keys.

Point system

The following actions gain points:

1. Ball collides with Bumpers
2. Ball picks up coin
3. Ball picks up all coins before they respawn

There are also multipliers gained for:

1. Time since last ball was deleted (global multiplier)
2. Amount of times the ball touched bumpers before being flipped again (counted for each ball separately)

Abilities

There are 2 pickups:

Multiball:

After a short time 2 new balls are spawned on the balls previous location. These are then given a random velocity, so they move in different directions.

Force up:

The base force, which all forces are multiplied by, is 1.25 instead of 1 for some time.

Implementation of criteria

This is a longer version of what can be read in the repositories readme.

0. Units and Position

I have chosen the diameter of the pinball as the 1, since the ball is the object that interacts with all other objects, and the most important object when it comes to spacing, for example the spacing between the flippers.
0 is located right below the tip of the flippers in reverse flip position, as there is no way to save the ball from that point on. This makes programing easier, there is no need for a "Death Zone" that is programmed, if the ball can't be interacted with when its Y coordinate is below 0. (Main.ts:145)
The X-Axis' 0 is in the middle between the flippers, simply to make symmetry easier.

1. Hierarchy

Scene
- Arena
  - Balls
    - Ball
  - Spring
  - Barriers
    - Case
      - Top
      - Bottom
      - Left
      - Right
      - Ground
      - Glass
      - BallguideLeft
      - BallguideRight
    - Corners
      - BotLeft
      - BotRight
      - TopLeft
      - TopRight
    - Pyramids
      - Pyramid1-3
  - Flippers
    - LeftFlipper
    - RightFlipper
  - Bumpers
    - Bumper1-5
  - Pickups
    - Coin
      - Coin1-5
    - MultiballAbility

1. Every Object, aside from the spring, is categorized and easy to find. The spring does not necessarily need to be categorized; it is only used to check whether the ball is inside of the “launch tube”.

Initially it seems having a single Ball inside the Balls node is unnecessary, but doing this simplifies the work needed to implement the Multiball power up. (Main.ts:138)
2. Since the Coins 1-5 are in their own node, it is easy to check whether they have all been picked up and the bonus points are earned. (CollisionHandler.ts:99)
3. After the Arena was built upright during development, all that was needed to give it the same angle as a real Pinball machine, was to rotate the Arenas ComponentTransform. This makes proper positioning of objects easier, both before and after the Arena was rotated.

2. Editor

The Arena is built using the Editor. This is much easier than using code if one builds something that is not symmetrical. But it's still fairly simple when making a symmetrical environment. That is why all ComponentTransforms are added in the editor by hand, as they were used to build the environment. ComponentRigidBody on the other hand is easier to use in code, because I have several of the same object in different places. That way adding colliders by code and having them be the same is easier than doing it by hand. Testing and changing values is also faster using code.
Furthermore, objects of variable number, in this case the pinballs are also better added and removed by code, rather than the editor.
The same is true for the power ups and coins.

3. Script components

I used script components to handle collisions. In this project script components were useful, because I was able to give the Bumpers, Coins and other Pickups a Scriptcomponent with the objectType parameter, which handles the collisions, rather than coding each EventListener separately. This also spared me from having to write a class for the bumpers, which would have been unnecessary, as their numbers don’t change, they don’t move and have no need for special attributes.(CollisionHandler.ts)

4. Extend

I extended the Ball class from f.Node. It was in that way useful to me, as the pinballs are objects of variable number and they can be generated more easily, when they are their own class. Furthermore, multihit combos can be tracked for each ball, as they all have their own multihit-attribute.
Balls are also created in different places in the code, the Main.ts as well as the CollisionHandler.ts. So, having Ball.ts is much easier than coding everything twice.
The same is true for the power ups and coins.(Ball.ts, Coin.ts, Power.ts)

5. Sound

In this project, the user's perspective always stays the same, therefore sound always comes from the same direction respective to the user's perception. The placement was simply done on an object that can quickly be reached codewise, when sound needs to be played.(CollisionHandler.ts:88, Main.ts:73)

There are 3 different sounds, a "pling" to indicate a bumper has been hit, and points are gained, a coin sound, similar to Super Mario, to indicate a coin is picked up, and one to indicate a power up has been picked up.

6. VUI

The interface shows the player how many points he has earned, as well as how many lives he has left and how high the spring force is. (GameState.ts)

7. Event-System

The event system is mostly used to handle collision of the balls with pickups and the bumpers. The event system triggers, when an object collides with an object that has the CollisionHandler script component. It was useful, because that is a more efficient way of checking for onetime collisions, whereas continuous collision is better checked inside the update function. (CollisionHandler.ts:83, Main.ts:139)

8. External Data

Two files are used for external data, one balances the point system: (CollisionHandler.ts:78)

- bumperValue determines how many points the bumpers are worth.
- coinValue does the same for picking up a single coin
- coinTime determines how much time in seconds the player has to collect all 5 coins, before they respawn

The other has parameters used in physics: (Ball.ts:33)

- ballWeight determines the weight of the ball.

9. Light

Everything should be nicely visible, and importantly, not confusing, therefore only 2 lights are used. One ambient light for a basic light level, as well as one directional light to make sure shadows are visible, and the player can recognize the shape of objects.

A. Physics

Every object has a rigidbody, and aside from the ball, which is dynamic, all are static. forces are used on the ball when shooting it from the spring, as well as on the flippers, so that they are stronger (especially with extra force pickup). Collisions on Bumpers for points. The coins and the powerups are triggers.