Part 2 Explanation

Aayush Shrestha

1+0+0+7+8+0+2+5+5 = 28 = EVEN

Object Pooling:

I implemented object pooling by creating a script to hold all the ducks in the scene. Text

Description automatically generated

I’ve made it a singleton as I only want one duck pool and at the start of the game, I instantiate the max amounts of ducks I will need (10 ducks per round). As it is not possible to have more than 10 ducks in a scene at once (since they can also escape) I will not be needing more, but to allow for future changes and expansions, if I need more ducks later, I’ve included a way to allow for that as well.

How my pooling works is:

I create the max number of ducks at the start, and as the game progresses, anytime I need to instantiate a duck into the scene, I will call for the GetDuck() Method. This will return a duck from the queue of ducks sitting idle and bring it into the scene.

If I wish to kill a duck, I call the ReturnDuck() method, which takes in the duck in question and returns it back into the pool, letting it sit idle until the next time it is called.

How it optimizes the scene:

Using object pooling here makes it so I do not need to instantiate new objects in run time allowing for a smoother gameplay and no lag spike every time I wish to instantiate a duck.

If I were to not use pooling, The game would need to destroy and instantiate an object during runtime, whenever I needed a duck. This would be very costly, and lag spikes can be seen later in the game when we need to destroy ducks frequently and instantiate them frequently

Profiler:

Graphical user interface

Description automatically generated with low confidenceAs we can see, at the start the memory jumps up as we instantiate a large volume of objects.

Graphical user interface, application

Description automatically generated with medium confidenceBut during the gameplay, it is smooth.

Timeline

Description automatically generatedEven as we instantiate an object, it stays smooth, no jumps and no lag.

Command Pattern:

How the pattern was implemented:

Text

Description automatically generated

An abstract class Controller command was created so two subclasses (inverted and normal) could be derived off it.

We use this pattern in the player’s script as we are controlling the player’s input with it.

In the player script, we check if we should invert the controls or not, and if we should, we use the invert class’s control, and if we should not, we use the normal class’s control instead.

How it works:

The score manager also keeps track of how many ducks were missed on top of how many ducks were hit and other round details. This lets us determine if two ducks were missed in a row. If it were missed in a row, we invert the controls (use the inverted control method).

Question 7.

Management system in use: score manager.

Score manager was implemented with a singleton. I create a score manager script, turn it into a singleton by making sure that script is the only one in existence in the scene



Text

Description automatically generated

With this, I have various public methods that I can call from anywhere in the scene by ScoreManager.instance.methodName();