DUCK HUNTER

A networked game developed by Pablo & Aaron

1.0 Description of the game

“Duck Hunter” is a combination of two classic games you can find in any funfair. The main objective of the game is to hit as many plastic ducks (yellow and orange blocks) and cans (cylinders) as possible.

You will compete with another player using networking functionality, and the one who reaches 100 points wins.

Related to the point system, the cans will have a value of 10 points, the yellow ducks will have 2 points and the orange ducks will have 5 points.

Ducks hit will disappear and cans will fall as you will expect on a real funfair game. The difference is that cans will only spawn at the beginning while the ducks are going to spawn until the game ends.

2.0 Development plan

The project schedule is going to be the next one: We are going to set the tasks in which we are going work each day of the week.

- **Monday 3rd**

Aaron & Pablo: Brainstorming. Try to look for a reachable objective and a game that could fit with the requirements. Start writing a draft for the Game Document Design.

- **Tuesday 4th**

Aaron: Player logic (behaviour and pc and mobile input system) and its network implementation.

Pablo: create each type of duck and implement their functionality. Create the pooler script which let us optimize the game.

- **Wednesday 5th**

Aaron: Spawning positions for players. Lobby structure and options and transition between lobby and game.

Pablo: Spawning and Unspawninng duck logic

- **Thursday 6th**

Aaron: Implement the player bullets logic (bullets behaviour, spawning and interaction with the enemies) and the score system with network functionality. Victory conditions and Game Over state

Pablo: Implement the pooler and the duck spawner with network functionality.

- **Friday 7th**

Aaron & Pablo: create a test scene with all the functionality of the game to check that everything works properly or fix the bugs or error that maybe appear.

- **Saturday 8th**

Aaron & Pablo: Setting of the final version of the game scene. Testing and balance of the game. Last network tests and fix possible bugs.

- **Sunday 9th**

Aaron & Pablo: Export the final version of the project and deliver it on Inspera. Make the individual reports.

3.0 Optimization

One of the requirements in order to complete this assignment is to optimize the game as much as we can. To reach that optimization, we are going to follow some steps we found in the official Unity webpage.

3.1 Summary of optimization for mobile games

Each mobile device is different, that is why we should try to optimise our game in order to be able to run it in every mobile platform.

We are going to see some of the general patterns or ideas generally follow to optimise a mobile game.

3.1.1 CPU and memory optimization

One of the most popular is the object pooling. The object pooling is a pattern which allows the game to reuse the game objects instead of instantiating and destroying them. This is used very frequently with game objects that will appear a lot in the game, like enemies or bullets.

Other recommendation in terms of saving in resources is avoid a reiterated use of expensive instructions like Physics.Raycast(), FindObjectsWithTag() or Mathf functions and try to use them only on Start methods or in punctual occasions.

The last thing usually recommended is related to sprites animations. The point is: is better to have a particle system which runs an animation than a script which does that functionality for every single object.

3.1.2 Rendering optimization

In terms of rendering, one usual pattern used is the *shadowgun*. This pattern consists in baking lighting detail into the texture instead of giving her some solid definition.

The real-time lighting implies higher quality, but the performance gained from the baked version is massive. To carry through this pattern we had an editor tool called "Render to Texel".

This tool does all the light and colour calculations in the editor, before the game starts, letting us to create a cool graphics for our game and implement them in a very efficient way.

The other idea commonly followed is to use different methods to render different light frequencies. For example, baked vs dynamic or per-object vs per-level. This is the way to render full bodied images on limited hardware.

3.2 Input in Duck Hunter

The controls of our game are going to be simple. We could move the player camera (view) dragging the finger along the screen with an inverted control for the vertical movement. We have also the possibility to use the mouse to move the camera or the view of the player for the Pc built. It has a similar functionality, but in this case by pressing the right button of the mouse instead of touching the screen.

Related to the shooting mechanic, we are going to have a button on the down right corner. Every time the player presses the button, he will shoot a ball.

3.3 Memory optimization in Duck Hunter

To optimise the memory usage in our application we are going to apply some patterns that help us to manage this objective.

We are going to create a pooler to instantiate all of our prefabs (bullets, different types of ducks) at the beginning of the game and reuse them while the players play. This pooler let us to save on CPU resources and memory because we could avoid to use instantiate() and destroy() functions every time which are very expensive in terms of means.

We are going to throw expensive instructions like FindObjectsWithTag() or GetComponent() only in the Start(), Awake() or OnStartServer() methods in order to execute them only once.

Related to Mathf and Physics functions (like Mathf.Log() or Physics.Raycast()), we will not use them in our app because we do not need their functionality to carry through our game. In this way, we are going to obtain a more efficient and fluent game.

3.3 Graphic optimization in Duck Hunter

We are not going to use art in Duck Hunter, then we do not need to do optimizations in this field.

4.0 Network communication

In this project, we are going to use Unet, which is the official unity networking functionality.

We will create a particular lobby room, which is going to have a max size of 2 players limiting the number of players that can join the room. The game could start with 1 or 2 players joined.

To build the data communication between the server and the clients we are going to use Command and RpcClient functions.

Command functions are sent from game objects on the client to game objects on the server. Those type of functions run on server when they are called from clients.

ClientRpc() functions are sent from game objects on the server to game objects on the client. Those functions can be sent from any object on the server with a network identity component.

The pooler objects (cans, ducks, bullets) have a very low network send rate in order to save resources and make our network communication more efficient. We are able to do this improvement because each instantiates in each scene.