Report

Game Design –

The game is a multiplayer snake-like game with a competitive twist. Players control a character that grows in length by consuming objects scattered around the map. The game involves dynamic player interaction, including collisions between players that determine a winner, and a loser based on the player's character length, but the game can also be played as a co-op game where players try to get as big as possible without hitting each other.

Network Overview –

The networking architecture of the game is built using Unity's Netcode for GameObjects (NGO). The game employs a client-server model with a server-authoritative approach, where the server is responsible for managing the game state, handling player actions, and ensuring synchronization across clients

Server-Authoritative Architecture

- **Movement and Interaction:**
    - The server controls the authoritative movement of players. The player movement inputs are captured on the client side but processed and validated on the server side. This ensures that all clients see consistent and synchronized player movements, preventing cheating or desynchronization.
- **Collision Handling:**
    - All collision detection and the resulting game logic, such as determining the winner in a collision between two players, are handled server-side. This centralization ensures fairness and consistency across all clients, as the server dictates the outcomes of such interactions.
- **Food Spawning:**
    - Food objects are spawned and managed by the server. The server periodically spawns new food objects, ensuring they are synchronized across all clients. When a player collects a food item, the server handles the despawning and updating of the player's length.

Client-Side Operations

- **UI and Feedback:**
    - Clients receive updates from the server about their current state, such as when they eat a player or when the game is over. These updates are sent using ClientRpc calls, ensuring that each client gets the correct feedback based on their actions.

- **Length Synchronization:**
    - The length of the player character, represented by the number of tail segments, is synchronized between the server and clients using NetworkVariable. When the player's length changes on the server, this change is automatically propagated to all clients, ensuring that the visual representation of each player's length is consistent across all clients.

Object Pooling

The game uses an object pooling system to manage frequently used networked objects like food and tail segments. The NetworkObjectPool component handles the instantiation, reuse, and despawning of objects, optimizing performance and reducing network load by minimizing the number of networked objects that need to be instantiated or destroyed during gameplay.

Challenges and Solutions Implemented –

- The game was originally a two-player split screen experience, so it was a great learning experience to try to convert it to a networked multiplayer game. Because the player character had to be contained within a prefab, I learned how to design a player class without adding dependency on objects in the scene.

- To improve performance and reduce network load, I implemented object pooling for frequently used objects like food and tail segments. This added another layer of complexity, particularly in managing the lifecycle of these objects across the network.

- The initial challenge was understanding the fundamental concepts of networking in games. It took time to grasp how different architectures impacted gameplay and player experience. It took a lot of trial and error to figure out the perfect balance between handling data and actions between client and server.

- Debugging issues in a networked environment was much more difficult than in a single-player game. With the help of some external packages, this task became a little easier but still a time-consuming challenge.

Reflection –

- One of the most important lessons I learned was the value of iteration. Early in the project, I tried to design the "perfect" architecture from the start, but this approach quickly proved to be impractical. Actually implementing the features and seeing where things go wrong and then coming up with solutions to those problems was a much more effective way of development.

- The project led me to understand the complexity of adding networking features to an existing non networked game. In the end, I had to almost rebuild the whole game from scratch. Which leads me to believe that in the future, any game that will have a networking component down the line, needs to be built for it from the start.
- In the end, watching two separate instances of my game being synced across without any issues was immensely satisfying.