CS21 Project Proposal

March 28, 2022

Team Name:

The name of our team is Mark's Sharks.

Team Members:

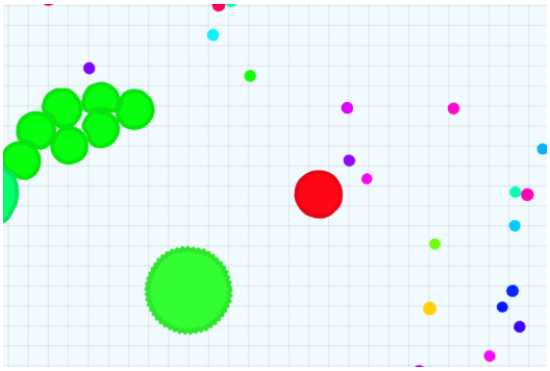
Our team consists of four members — Ankur Dahal, Ellis Brown, Jackson Parsells, and Rujen Amatya.

Project Description:

"SharkIO"

Using Python3, we will be creating an online game with similar game mechanics to the popular online games "agar.io" and "snake.io", which are online, browser-based games which allow for 1-20 people to play concurrently. Our game involves moving a blob/snake around, eating food to grow in size, and eating other players to grow in size.

A screenshot from the online games that are giving us inspiration for SharkIO can be found here.



A player will control a single blob, and can move around in 2-dimensions on the game board. There will be obstacles that the player should avoid in order to not lose their mass.

Minimum Deliverable:

Our minimum deliverable will be a single player game with randomly moving “players” that will start up when the player joins. We’ll use a concurrency based python backend with each thread representing a player. To consume another player, you must be currently larger than that player; to gain initial mass, you must eat randomly scattered bits. We’ll also introduce stage hazards that will damage you if you are larger than it. In our minimum deliverable, we plan to focus on just getting the game working, so animation will not be present in any place not explicitly necessary, and the players will stay the same size despite how much they’ve consumed (score will be written on the player’s avatar though and will be updated as the game plays on).

Expected technologies to be utilized:

- Python client-server model with the client using the Pygame library
- Web sockets for communication between the server and clients

Maximum Deliverable:

Our maximum deliverable/goal would be to implement the multiplayer mode, with up to 20 players, and our in class demonstration would allow every student in the class to play concurrently. We are aiming to add animations and graphics to show the scores of individual players, and also have a global leaderboard to display the current leader in the game room. We plan on allowing multiple rooms of the lobby to exist that are independent with their own players. If time permits, we plan on allowing for spawning processes across multiple hosting web servers as well.

First Steps:

The first steps would be to focus on the basics and implement the required classes and set up basic server states. We would make choices about the concurrent models in our game, and the local / shared data they contain. To do this, we would sketch out the architecture of our application and create a model of the data flow for all entities involved.

Potential Problems and Questions:

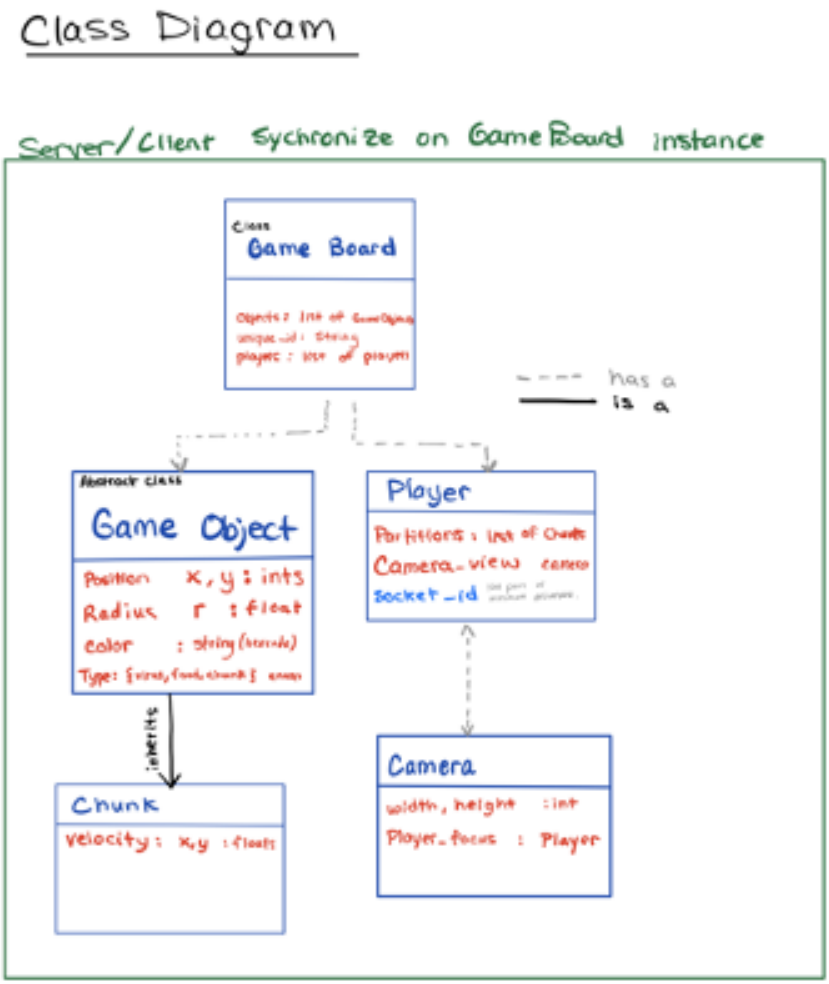
- Using web sockets for multiplayer mode
- Python GIL for scaling into players above ~10
- Shared gameplay for more than 1 user
- Animations and rendering graphics
- Any unforeseen technical issues as a result of using web sockets for communication

Changes in the design:

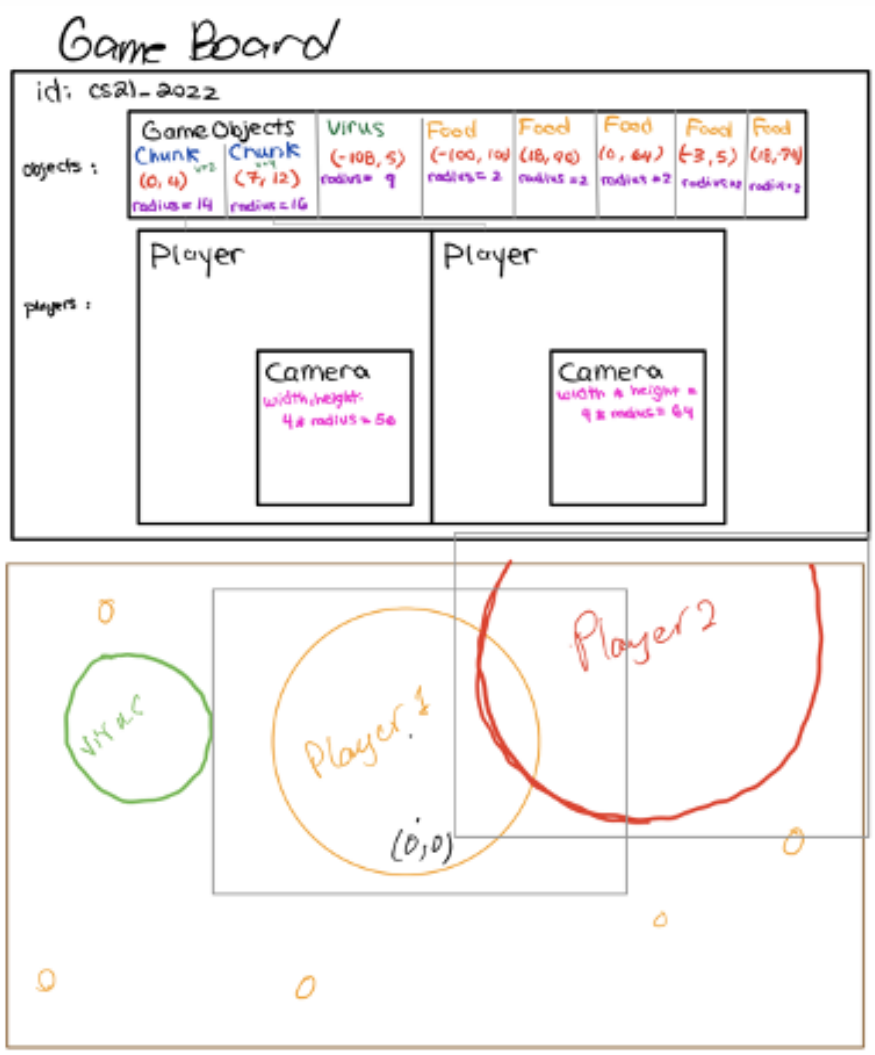
Our design decisions evolved as we tried to write our first set of classes. In our initial proposal, we mentioned that we were going to use a Python backend and use a web server to implement our game. However, after analyzing the options, we deviated towards using the Pygame library to make the game because of its easy graphics rendering API, support for sounds, and overall ease of use of the library and the vast documentation available for it.

Design decision:

We will use a Server holding an array of game objects and an array of players, and the ground truth is on the server side. Each client only gets an updated list of deltas, and then sends movement info to the server, and then the server will handle the collisions and update required state. All communication between server and clients is done via web sockets. Client side must recreate the full game state from only these deltas, and each client player instance only renders the view seen by its corresponding camera instance. The concurrency portion of the project is seen through multiple threads for clients, and a thread for the server. Each client has a process spawned on its local machine, while the server has a thread that represents each client spawned locally, and another thread for tracking the game state overall. Each thread has a one-to-one correspondence with the client process it represents on the server side.



Object diagram:



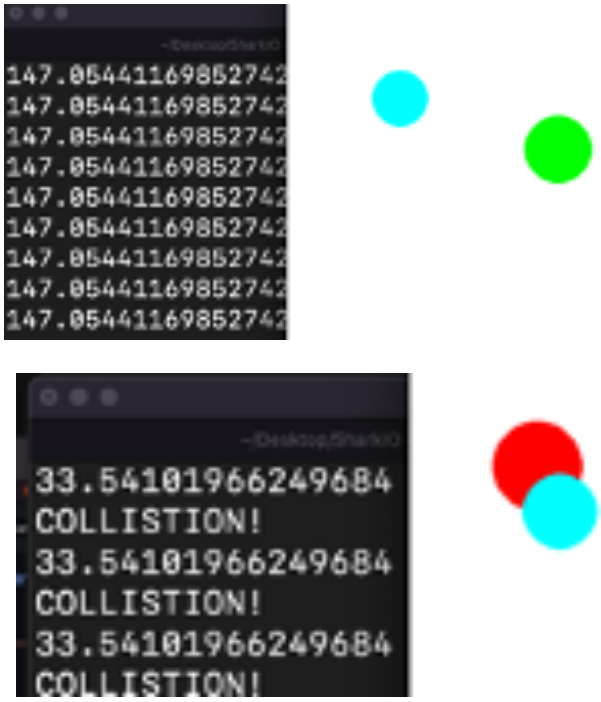
This object diagram captures a moment in the game when two similarly-sized players are overlapping. The game board contains other game objects like a virus, and food particles as well.

Progress:

We have started implementing base classes like Game Object which are fundamental to the interaction between several entities in the game.

We have also begun working on detecting collisions, which works by using the positions of the objects, the distance between them, and their radii to detect collisions.

We have a small working demo of a simple prototype where one can move a player around a canvas and collect food.



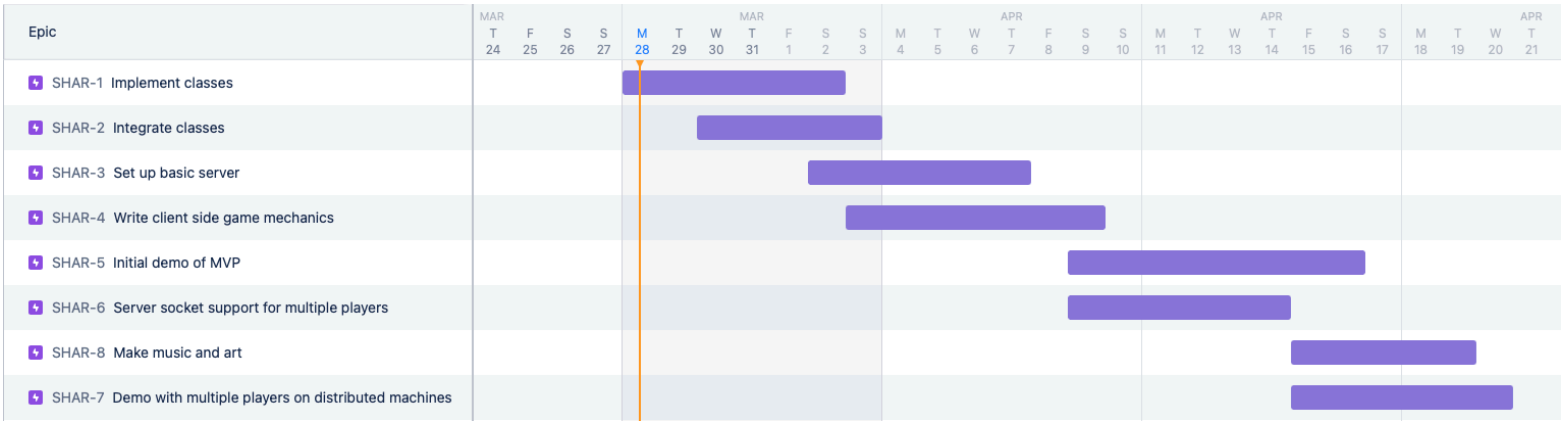
Here are two snippets from our first demo in Pygame, with a terminal output of collisions of the two circles. The player circle is the green one in the first diagram, which turns red as soon as it collides with an enemy, which is blue in this screenshot.

Division of labor:

We anticipate we will have sufficient time with 4 individuals if we meet during class in addition to 2 other synchronous meetings either in person or over zoom each week, totaling 6 hours per week of labor per person not counting unanticipated stretch periods. Therefore, as we plan to all work together synchronously on each ticket. This will enhance both overall team learning and cohesion.

Planned Roadmap:

The roadmap is illustrated in the Gantt chart below:



To summarize, we plan on finishing up implementing and integrating classes by the start of next week. While we are doing so, we will divide work so that some members can concurrently work on setting up the server and client side application. This is expected to take at least a week, and by the end of the second week, we expect to have a single player prototype of the game. While we test the game, we plan on integrating multiplayer game mechanics to our game, and work on it for one more week before we polish the details by making art and music for the game and finish up the project with a working multiplayer demo.