Kevin Guo
Cedric Kim
3/10/16

Pygame Networking within a game

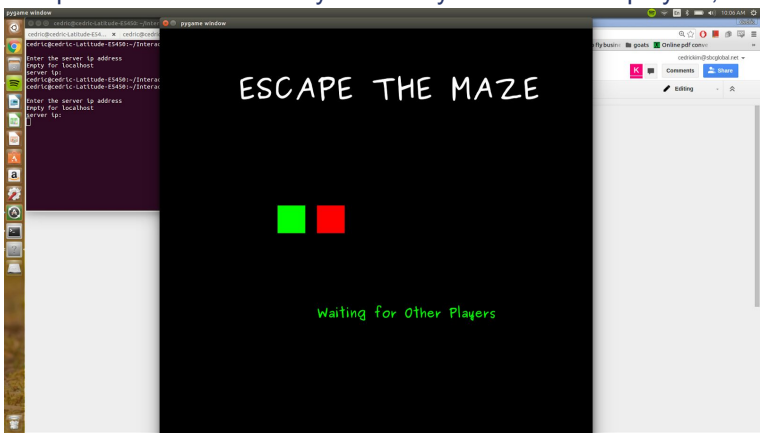
Project Overview

Our project is a game that incorporates networking. The game can have any number of players within the server. The players are spawned into a maze, with one person randomly picked as a killer. The player's goal is to reach the exit, if a civilian, or to kill the other players, if the monster. This project implemented PodSixNet libraries in order to create a network between an unlimited number of computers.

Results

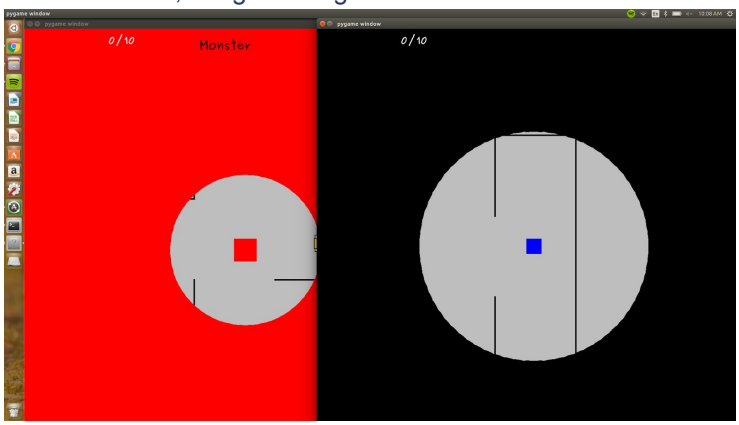
Our project successfully implemented a networked system, where the computers can communicate to each other over a network. We created a game with collisions, and multiple user input event cases.

This picture shows the lobby in which you can see the players, and where the players can ready up



Once the players press space to get ready, the game enters the story mode, which displays the background story (different for the monster and other players). This screen also displays the keys in which the character can move (WASD)

After a set time, the game begins



The player that was randomly selected to be a monster has a red screen, and can walk around. The players attempting to escape as well as the monster have a Fog of War, which limits your vision around you. The monster's FOW grows overtime as the game progresses.

The Monster wins once he kills the other players (walks on top of them).
The other players win once they collect the randomly generated scrolls and reach the exit (which appears once the player is in close proximity to it)
After the players win, or die, they enter the spectator mode in which they can view the other players run around the maze.

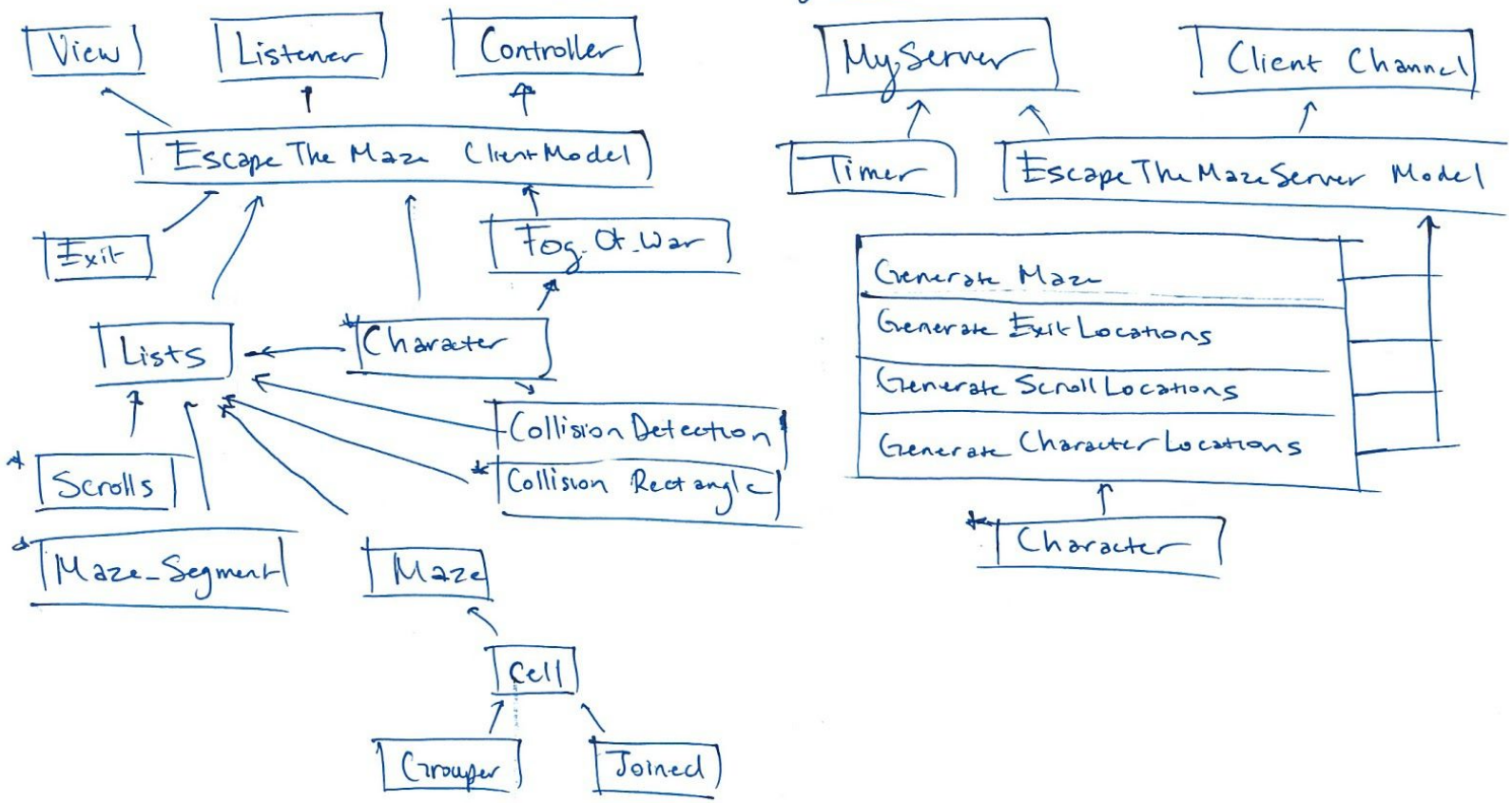
Implementation

Our system architecture implements a modified Model-View-Controller. Using the Model-View-Controller we learned about in class (brick breaker), we were able to modify the model to include a maze and have different keyboard inputs. We had multiple classes for the different objects within our game in order to store all our attributes. We created a Collision class which implements the different collisions (2-d collisions, collisions between different objects). At one point, our system began to become so complex (lots of things in the model class) so we decided to create a new class called Lists, which stores all the different types of lists and does different kinds of list computations to fit our needs.

Our network involved having one instance of a server and an instance of a client for each player. In the server, we generated all the data such as the maze structure, random character locations, random scroll locations, which player is the monster, etc. We then sent this data to the players at the start of the game. Afterwards, the server simply serves as a facilitator for packet transfer. In the client's program, there is a listener class which has methods for retrieving data. The client also pushes data to the server, which then gets pushed from the server to every single client (including the original sender of the data). As a result, all players are constantly updated about other players' crucial information.

One major design decision was sending simple data. This helped to reduce the amount of time and amount of lag in the game. The client can reconstruct the data itself since that process is much easier than sending complex data over a server.

UML Diagram


View
Listener
Controller
MyServer
Client Channel
Escape The Maze Client Model
Timer
Escape The Maze Server Model
Exit
Fog Of War
Generate Maze
Generate Exit Locations
Generate Scroll Locations
Generate Character Locations
Lists
Character
Collision Detection
Collision Rectangle
Scrolls
Character
Maze_Segment
Maze
Cell
Grouper
Joined

Reflection

We had a relatively successful project because we completed our project to the goal we had envisioned and beyond. We had many ideas for the project that we wanted to implement, but realized that inevitably did not have time to do them all. However, many of these ideas were even beyond our original reach goals. If we were to continue working on the project, the main changes would be making our maze a combination of fewer rectangles to decrease lag. Additionally, we wanted to be able to use ray tracing to create a mouse-controlled flashlight feature instead of using the circular view.

Knowing how to run the network would have allowed us to make our program much more easily since we struggled with understanding how to efficiently send data. PodSixNet didn't have much documentation and most example programs didn't implement objects into the network, which we needed in our program.

We divided the work by working on different aspects of the program independently and then reconvened to create the whole program again. Occasionally, we would both work on the same component of the program if we were really struggling for it to work. The main issues were due to large restructuring of the program in ways that made it difficult to implement other people's work. In the future, large restructuring should be done together to make sure that no changes are made which may be potentially useless.