COP701 Assignment 2 Report

Aditya Mohan Mishra (2018ME10582)
Harsh Wardhan (2018EE30610)
Pushpit Srivastava (2018BB10031)

Initial Module Plan

Server:

-State Merger:
  A: Takes state from both clients merges them and returns new state to both
-Communication Module:

Client:

-Renderer:
  A: Tkinter
-Communication Module:

Common:

-Physics Module:
  A: Line Equation
  B: Collision detection

-Overall State:
  A: Both Player State
-State per player :
  A: Orientation of player block
  B: Location of player block
  C: List of bullet blocks
  D: Number of bullets with player block
  E: Points

-Balloon:
  A. Position

-Bullet Block:
  A: Origin of bullet
  B: Trajectory of line of bullet
  C: Size of bullet

-Game Mode:
  A: Velocity of bullets
  B: Regeneration Rate of bullets
  E: MaxBullets
  F: Balloon seed

Summary of Final Modules

The objective of the game is to maximise the number of balloons popped by a player. Bullets are shot at a constant rate by each player. The player can aim by left-clicking or right-clicking the mouse. Balloons are spawned periodically. The game ends in 2 minutes from the start time.

tick.py has methods to maintain and sync time for the game. GameMode class stores the parameters or “rules” of the game.

Circle (geometry.py) is the class used to implement balloons and bullets. Balloons and Bullets classes inherit the Circle class.

Each player’s state is stored in objects of the PlayerState class. This class stores the position, orientation, points and list of bullets fired by the player. The class has methods to shoot bullets and change the orientation of the player.

OverallState keeps track of the state of the game and all variables that are required. An object of this class boots the game by creating Players (using PlayerState) and also updates the state of the game based on its current state and user inputs from the players.

GUIManager inherits Tkinter.Window and handles display of all GUI elements. It inherits Frame objects like Scoreboard, Graphics, StartMenu, etc and displays them in a grid.

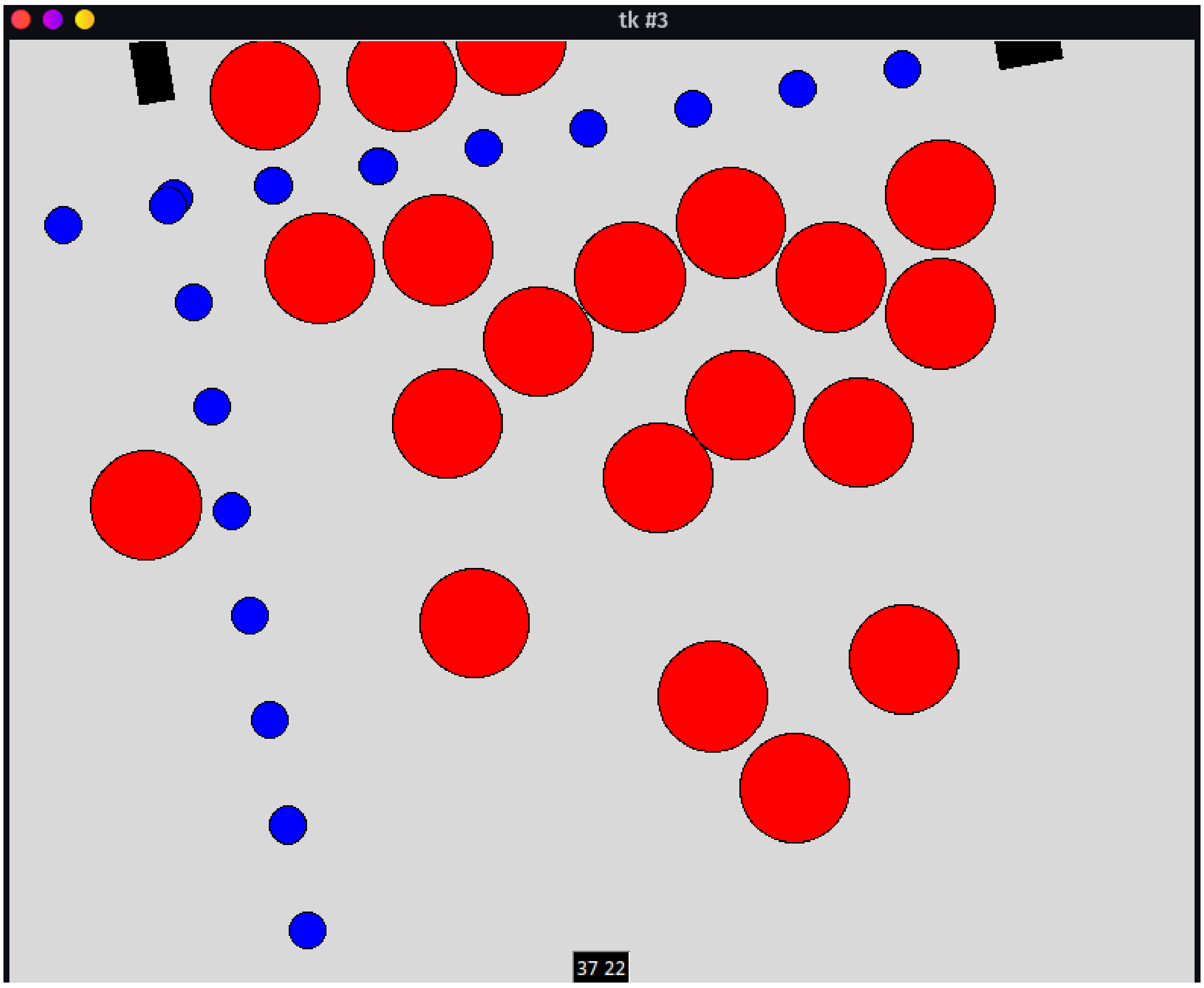
Architecture and Reasoning

OverallState class maintains the entire state of the game by housing the playerState(which houses the bullet locations of all the bullets this player has) and also all the balloon locations. This class also houses the function updateState() which takes two input arrays that are time stamped inputs made by the two clients. Periodically feeding inputs this function via threading each client maintains it state on local machine and also periodically via threading it sends its input list to server who takes the inputs of both clients gets the updated state via updateState() and sends the resultant state finalised by it to both clients for them to overwrite and set their final state from. This construction allowed us to limit the functioning and physics of the game to a single updating function and present a common library to be consistent between servers and clients. Our netcode follows a rollback architecture where the server sends their final state which partially undoes the state on the client. updateState() functions as the main engine of the game and is responsible for

updating gameState. Also this reduces the input/output between server and client as client only sends its inputs to server.
The GUI interface of the client only mirrors the state which is present on the client and is just a Tkinter canvas object drawing the current state.
Since the server is looking at the time stamped inputs of clients, the difference of system times b/w machines and the network delay was also considered to build up to this architecture where we finalised an exponential averaging algorithm to maintain the difference b/w server and client time and process operations accordingly.

Screenshots of the Game

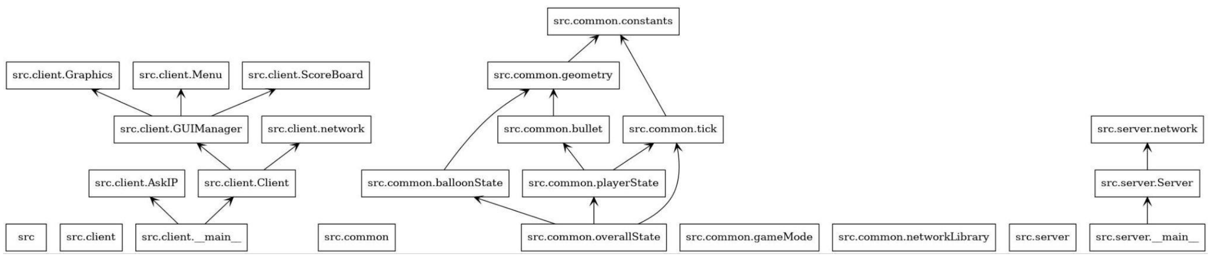
Gameplay image
Winning Screen
Losing Screen
Respectively

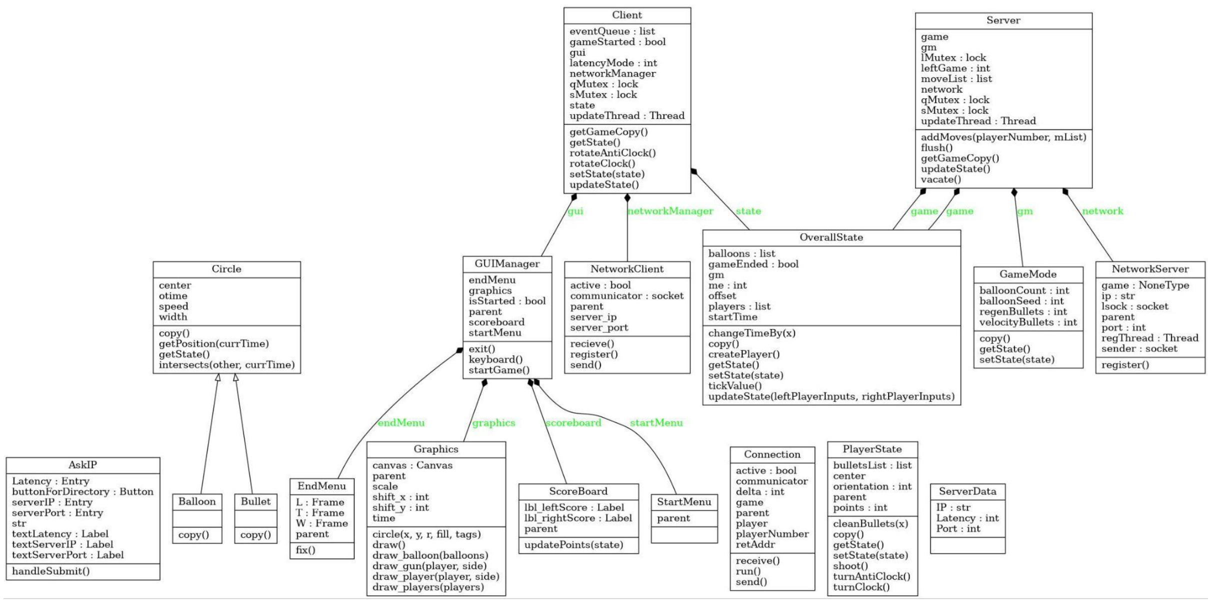


LUCKY TRY

Flowcharts

Inheritance Graph





Classes and their functionality along with objects UML