Part 1:

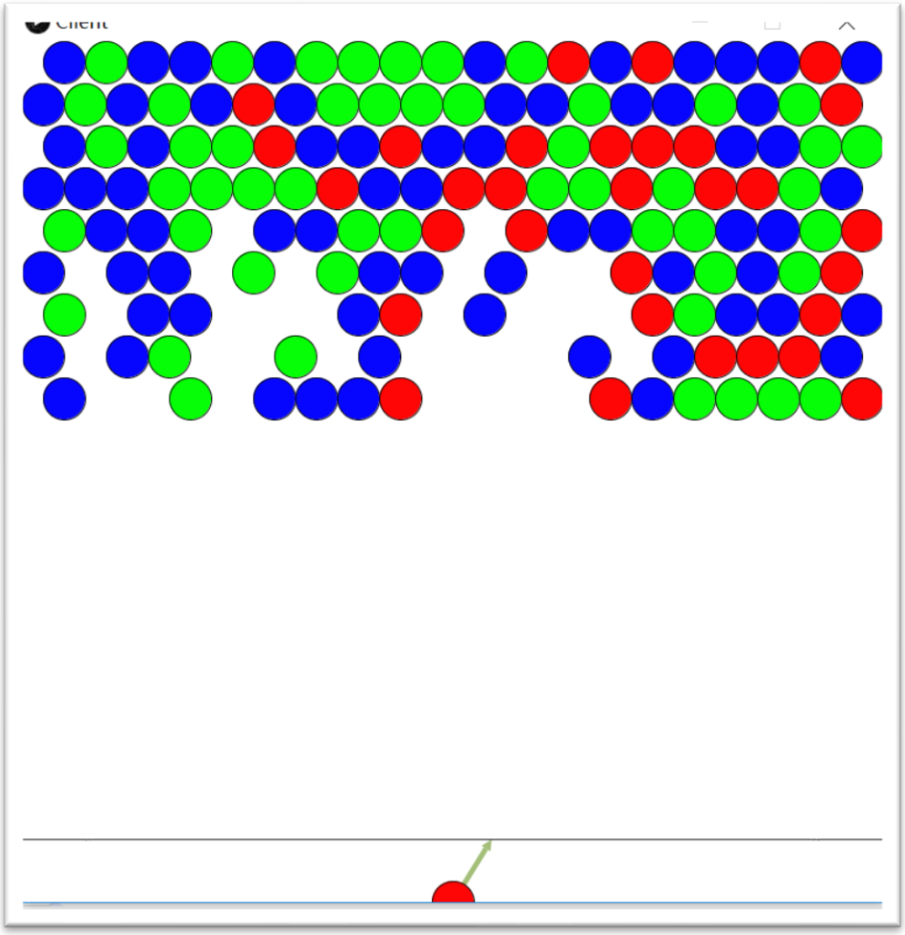
To incorporate “scripting functionality” into my game engine. I implemented a script manager using the javax.script package and referred to the demonstrating code given by professor David.

My script manager has the following functionality:

- bind arguments
- load script
- execution script

And in my game implementation. I use script (javascript) to modify the movement of dead zones and handle event where game characters collide with platforms.

Part 2:



In this part, I implement the “Bubble Shooter” game.

As I mentioned in the Part 4 (Reflection), I change a little bit about data transmitted between server and clients. Other parts of the game engine (script system, time system, event system and game object model) stay the same.

I create new “Movable” game objects like “Bubble”, “Bubble Shooter”, and “Renderable” game objects like “Dead line”, “color indicator”. All those game objects mentioned above use the basic game object model system defined previously.

New events like “Bubble Dead Event”, “Bubble Gone Event” and “Shoot Bubble Event” is created. To make the visual effects of bubble disappearing more obvious, I take advantage of the

event system. To be specific, when a group of bubbles are going to disappear, a “Bubble Dead event” which will be handled immediately is first enqueued. This event simply change the color of the chosen bubbles to white. Then a “Bubble Gone Event” which is designated to be handled one second later is enqueued. This event will remove the cluster of bubbles from the “game world” (i.e. a list of all the game objects). In this way, whenever bubbles are detected to be removed, they first turn white for a second and then disappear. I can achieve this in the game implementation all due to the delayed time processing enabled by event system

I used javascript to descript the movement of the bubble shot by the bubble shooter, which is enabled by the script system.

My “Bubble Shooter” game is fully functioned. There are 3 types of bubble in this game. After every 6 click, all the bubbles will go down for a level and a random layer will be added on the top. The criteria for dropping bubbles is three bubbles of the same color. And whenever the bubble has touched the bottom of the sketch (i.e. the dead line drawn on the sketch) or all the bubbles are dropped by the bubble shooter, this game will simply restart.

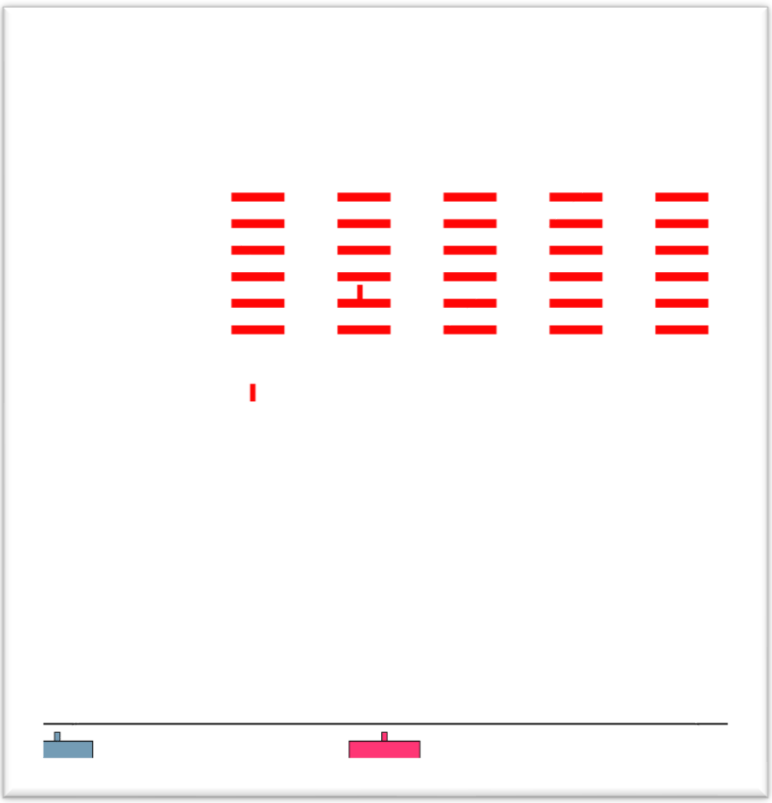
Server deals with all the game logic and data calculation and client is a dummy renderer.

About 57.8% of game engine code is the same with the platform game. So my game engine is highly reused.

For the algorithm part, when implementing this game, I mainly solve the following problems:

- where to place the new shot bubble? As bubbles are not placed randomly, the place where bubble lands have to be considered carefully.
- how to detect the connecting bubbles? I used depth first search in this part.
- how to detect dangling bubbles? Also, depth first search is used.

Part 3:



In this part, I implement the “Space Invader” Game. The logic in this game is simpler than the bubble shooter and I finished writing this game in less than one day. So you can see my game engine is heavily reused.

After changing the data form transmitted between clients and server in the Part 2, my network system works totally fine for this part.

Same with part 2, the components in game engine like script system, time system, event system and game object model stay the same.

In the implementation, I create the “Movable” game object “Space Ship”, “Gun Turret”, “Bullet” and “Renderable” game object “Dead line”.

In the event system part, I reuse the “Move” event from the platform game. Meanwhile, I create new event like “Turret Dead” event and “Turret Spawn” event. I used the evet system to make the visual effects of turret’s dying more obvious like the way I explained in Part 2.

Code similarity is about 69.3%.

In the gam play, whenever the spaceships touch the bottom dead line or turret destroyed all the space ships. The game will be simple restarted.

For the algorithm part, to solve the bullet collision problem (i.e. I only have one “Bullet” class, So it’s necessary to distinguish the bullets generated by turrets and spaceships), every bullet is assigned a Boolean called “friendly”. In every game loop iteration, only collisions between “Space Ship” and “friendly bullet” or “Gun Turret” and “unfriendly bullet” will be detected.

By the way, all the games I implemented are able to support multiple clients and a time. So they can be considered multiplayer games.