Zulu Team

CS619

10/12/18

BulletZone Design

1. **[3 pts]** Consistent and unambiguous functional requirements for both client and server. Don't forget to include the constraints on tanks and bullets as requirements for the server.

Up to eight clients may join the game at once.

The server must be able to receive events from the clients

The server must be able to interpret the info from received events

The server must be able to reply to the clients correctly

The server must reply within _ ms

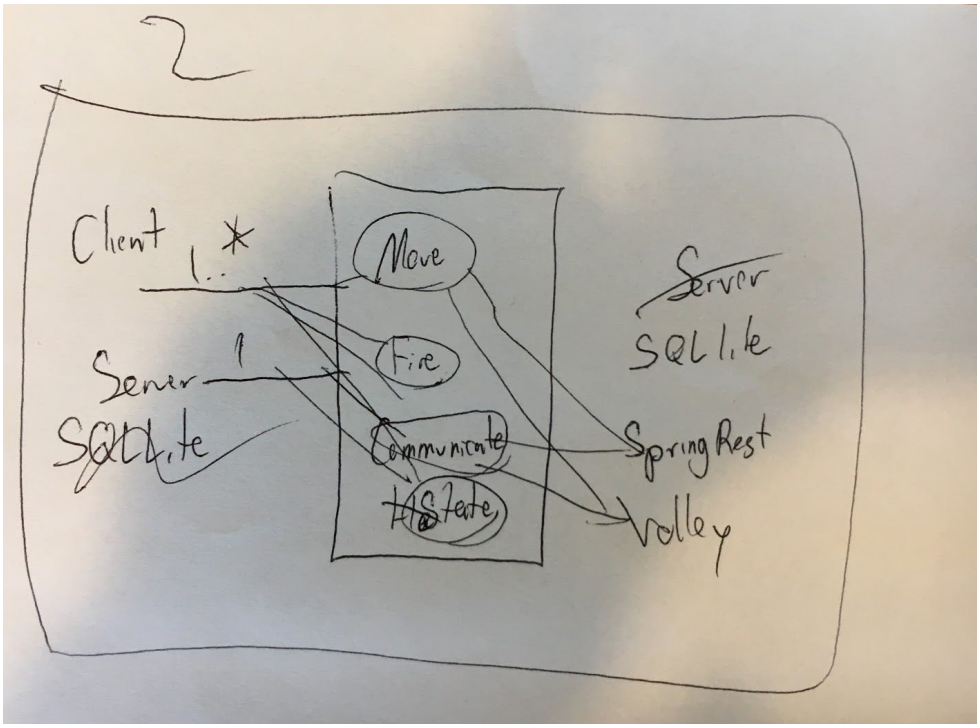
The client must save grid states to sqllite database

There must be a 16x16 grid representing the battlefield.

The server is responsible for managing the battlefield, game time, and enforcing the rules of the game.

The client is responsible for showing the game state to the player and provide a control interface.

2. **[3 pts]** Use-case diagram (showing actors and potential use-cases you have identified so far).



3. **[4 pts]** (At least) one main success scenario for (at least) one non-trivial use-case for the client app.

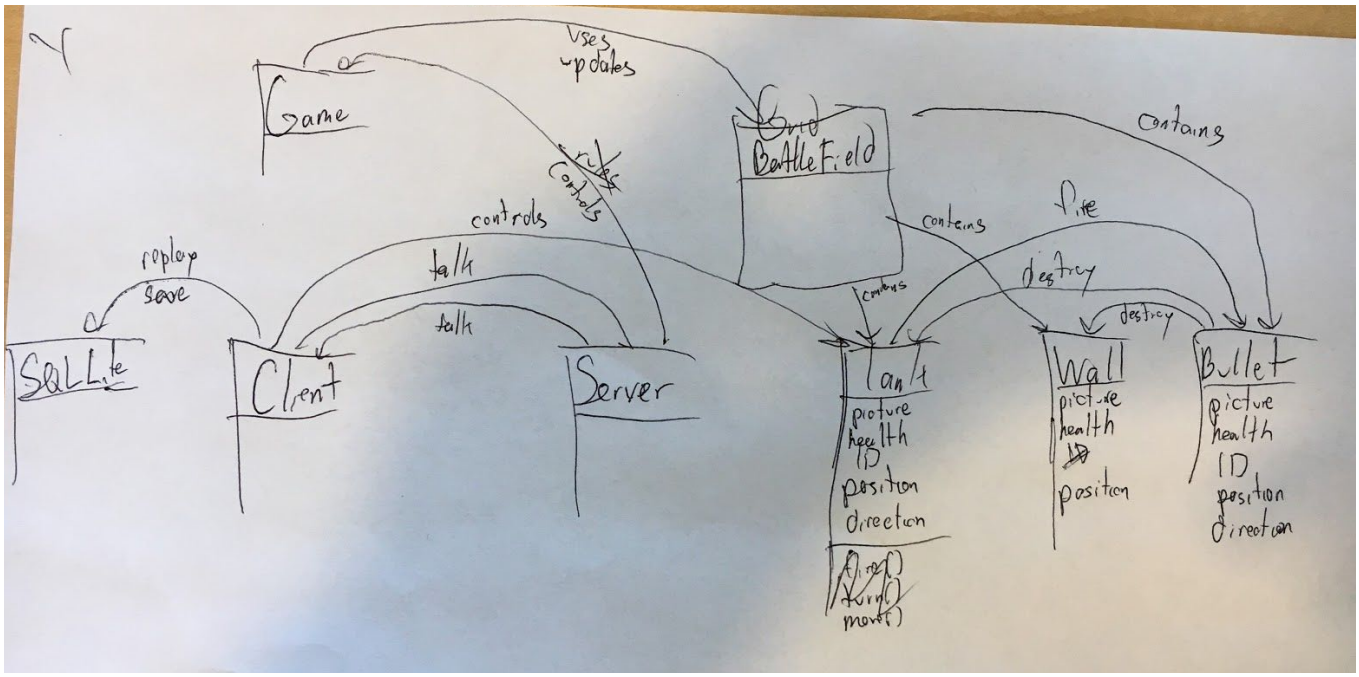
Player plays game

- Client requests to join game to server
- Server finds a game
- Server initializes tank
- Server sends back information to the client
- Client fires bullet
- Client sends “fire” request to server
- Server updates locations of tanks and bullets
- Server calculates collision with tank and bullet
- Server sends end message to other client
- Server sends win message to first client
- Client displays win to user

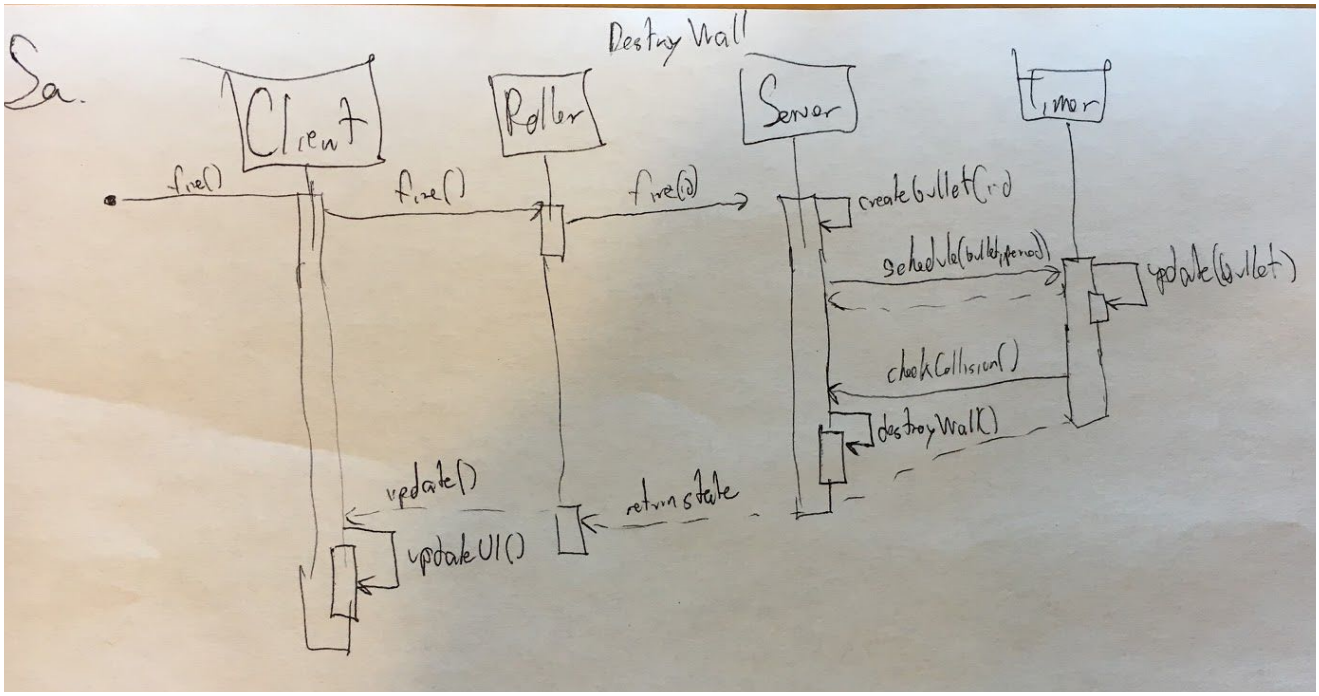
Precondition: client loads application

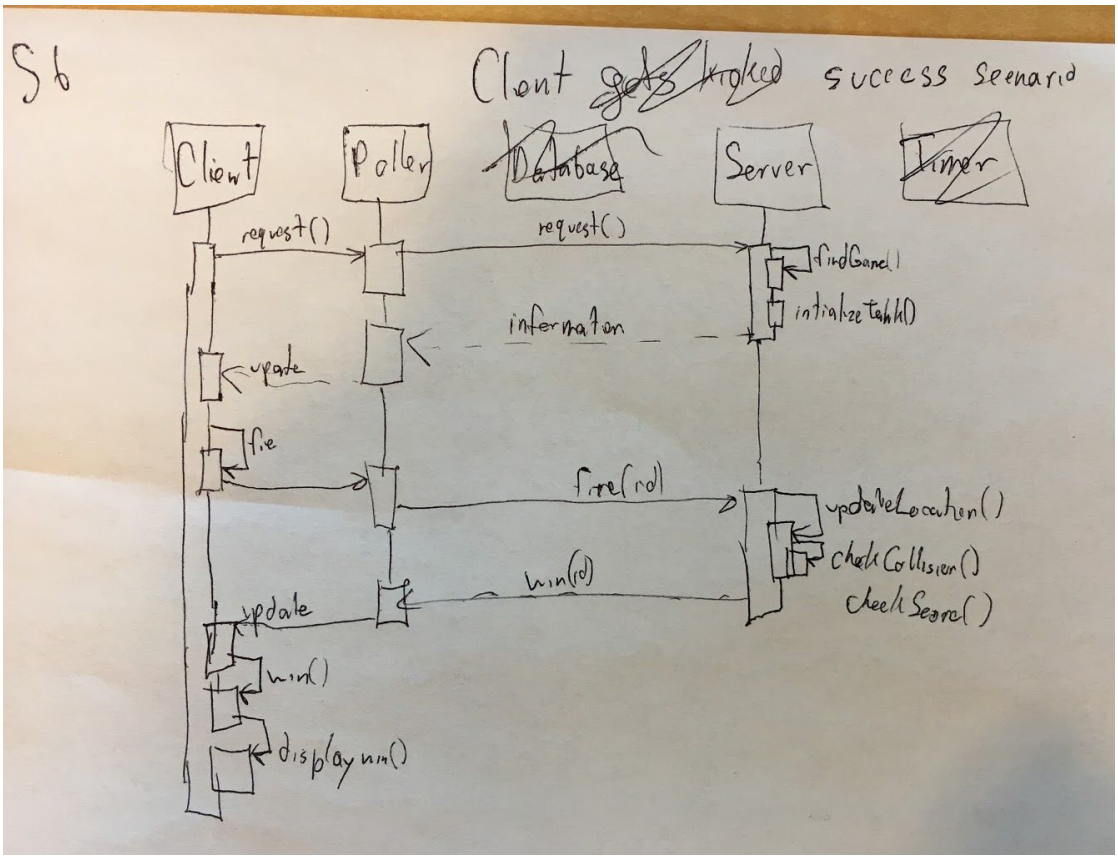
Postcondition: client wins game

4. **[3 pts]** Domain model (diagram of domain concepts you have identified so far, and their relationships).

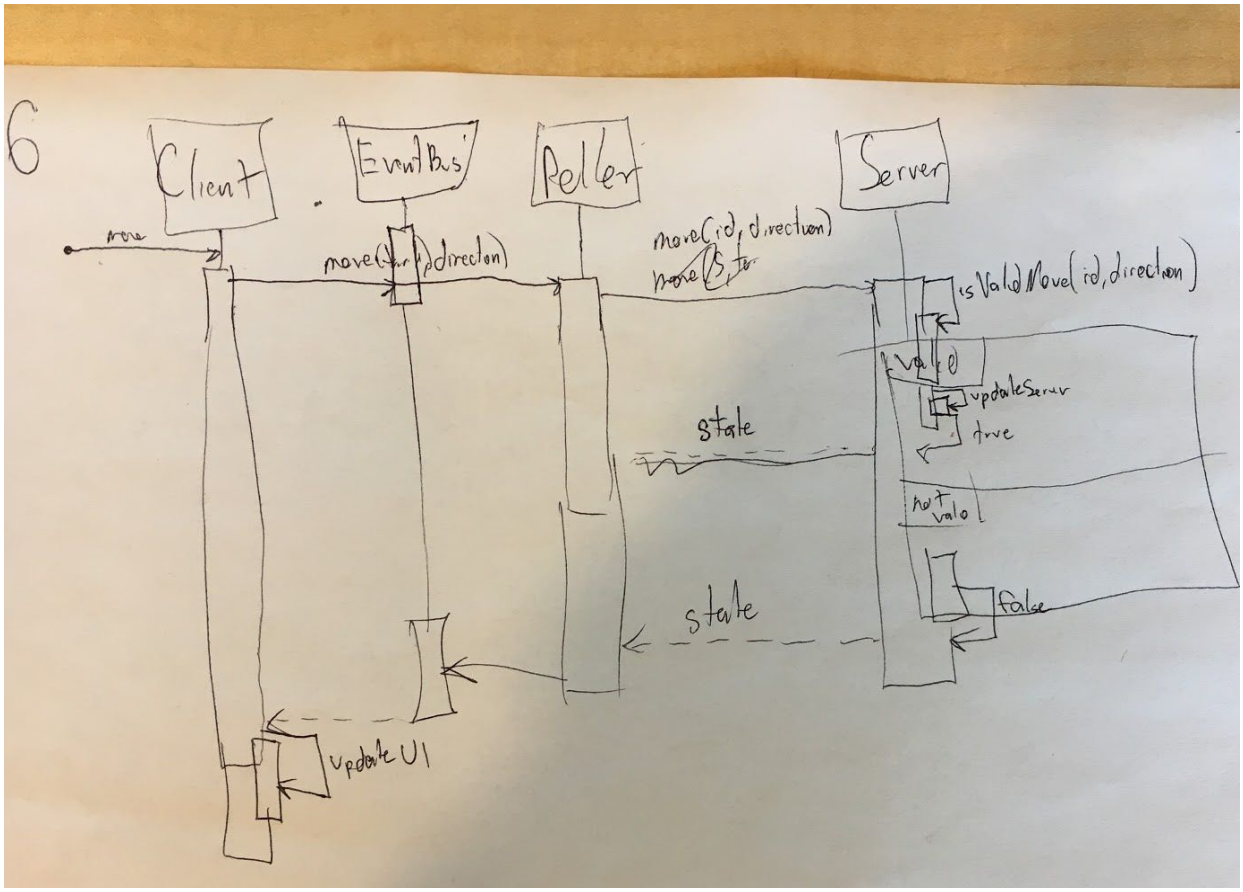


5. **[4 pts]** (At least) two UML sequence diagrams starting with a user action in the client, improved or different from what's in the assignment description. One sequence diagram should correspond to the success scenario you gave.

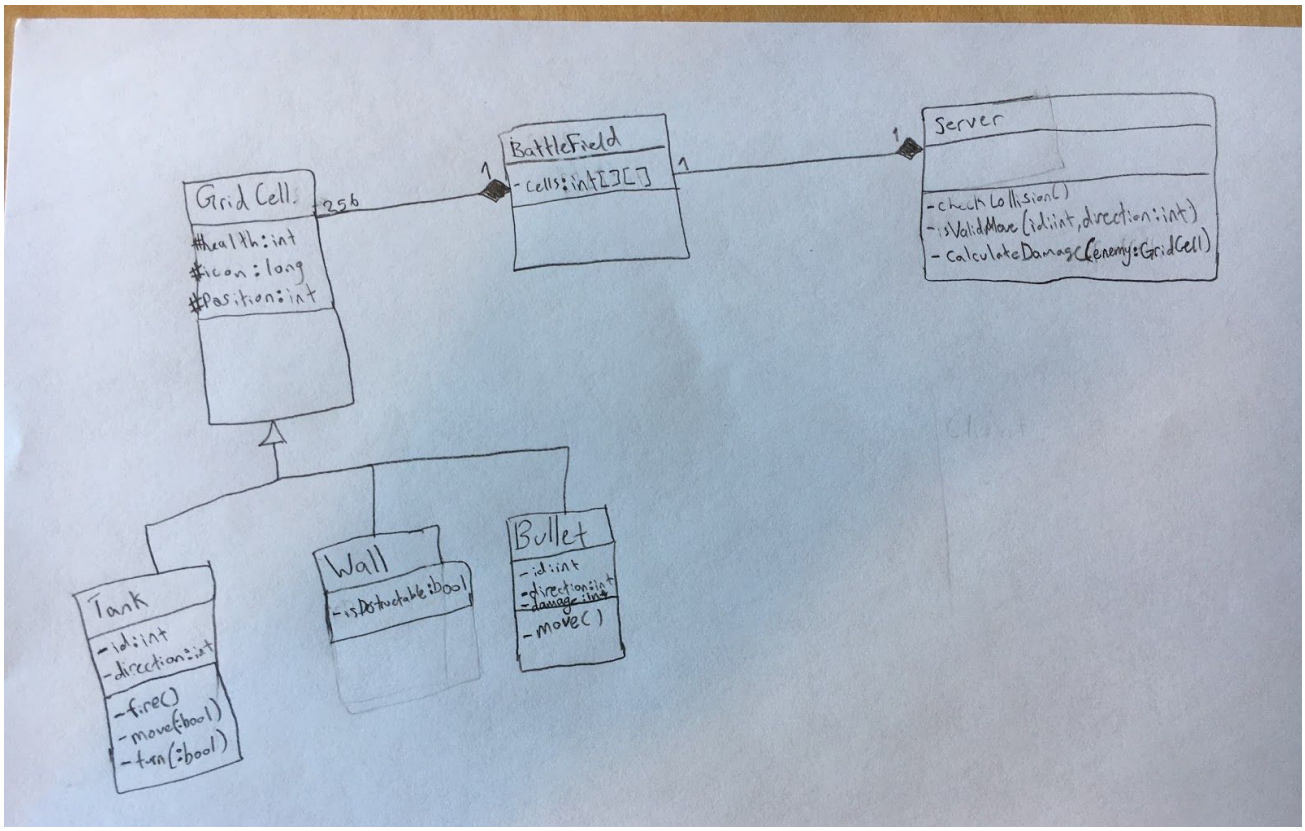




6. **[2 pts]** (At least) one UML sequence diagram illustrating how constraints on turning or movement will be enforced on the server, resulting in a different return value in server responses to turning or movement requests from a client.


6
Client
EventBus
Peller
Server
move
move(id, direction)
move(id, direction)
isValidMove(id, direction)
valid
updateServer
drive
state
not valid
false
state
updateUI

7. **[8 pts]** UML design class diagram(s) that supports joining a game plus (at least) one gameplay use-case... preferably should address the needs of the entire system



8. **[3 pts]** A brief description of what patterns you have identified so far as being useful for your project--indicate which classes will participate in each pattern, and what roles they will play in each pattern.

The initial design must be approved in order for the team to start project implementation, but you are encouraged to play around with the prototype code so that you can understand how it currently works. The design documents can include sketches at this point (pictures or scans of sketches can be submitted but should be of good quality), but realize that digital versions will be expected for the final documentation at the end of the semester. Your design should follow OO design principles discussed in class and design patterns should be applied appropriately.

The design document(s) must be uploaded to Canvas by the deadline, where I will be grading and giving feedback. It must also committed to gitlab.cs.unh.edu (Git) so that there is a record of the documentation that can easily be referred back to over the course of the project.

[10 pts] Each team will then present the design to the instructor by October 18 (during class time for most people) in Kingsbury W244. All team members must be present during the presentation for full credit. The design checkpoint and meeting account for 8% of the project total, and the grade is given as a group (no individual differences for this milestone).

8)

We've identified a few patterns in the design so far. The code will be designed based on the GridSim application, and will inherit some patterns from that application.

The factory pattern will be used to create all the objects of the grid based on a number.

The facade pattern will be used to simplify and mask some of the subsystems such as displaying the graphics with Android, and interacting with the storage database.

The observer/event bus will be used to communicate between subsystems in a simple manner.

The singleton pattern will be used to make sure there is only one factory, and one controller for some of the complex systems.