BulletZone

Baryte Team

Functional Requirements (Server)

Functional Requirements for Server

- Not Gameplay
    1. Accounts store information in a garage
    2. Server maintains board state of the game
    3. Server restricts board size to a 16x16 grid
    4. Server simulates destructible and non-destructible walls on the grid
    5. Server tracks game time
    6. Server manages player accounts
    7. There is only one server
- Gameplay
    1. Server evaluates restraints during gameplay
    2. Server detects hits on tanks
    3. Server limits tank movement to once every 0.5 seconds
    4. Server limits bullets from tanks to once in every 0.5 seconds
    5. Server limits only two bullets on the field per tank
    6. Server limits tank movement to one turn per step.
    7. Server denies sideways movement with respect to the tanks direction.

Functional Requirements (Client)

Functional Requirements for Client

- Not Gameplay
    1. Players can make an account
    2. Players can connect to the online server
    3. Players can login
    4. Players can view replays
    5. Players can change speed of replays
    6. Players can see the battlefield
- Gameplay
    1. Players can turn and move their token
    2. Players can fire bullets by shaking device or through features on the game user interface
    3. Players can turn and move to control the tank through the user interface.
    4. Various types of bullets can be fired by the player

Use Cases Diagram

Primary Actors

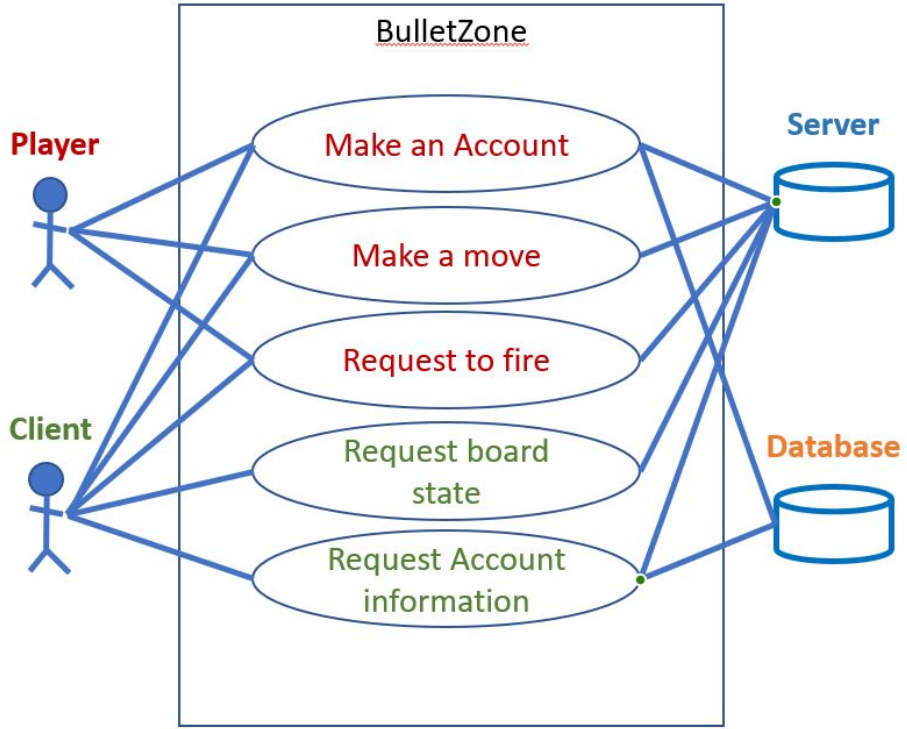
Player
Client

Supporting Actors

Server
Database

Goals

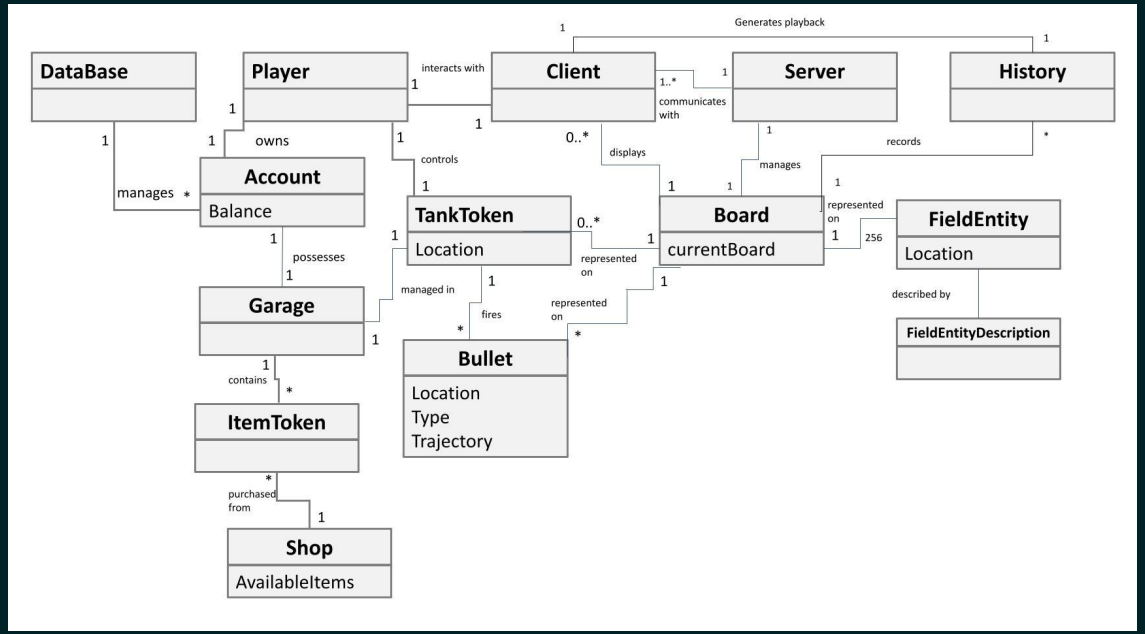
1. Make an Account
2. View Garage
3. Watch Replay
4. Buy an Item
5. Make a move
6. Request to fire
7. Request board state
8. Request account information



Main Success Scenario!

1. Player requests to fire (Use case)
    - 1.1. Player shakes their Client
    - 1.2. Client detects the shaking
    - 1.3. Client sends fire request to server
    - 1.4. Server receives the request
    - 1.5. Server checks board state of the game
    - 1.6. If two bullets from player exist on screen or has fired within .5 seconds
        - 1.6.1. Server denies the shot
        - 1.6.2. Server returns denial to client
        - 1.6.3. Client denies the request to fire and reports it to the player
    - 1.7. If one or fewer bullets exist from player and hasn't fired within .5 seconds
        - 1.7.1. Server accepts fire request
        - 1.7.2. Server returns success to client
        - 1.7.3. Server updates board state
        - 1.7.4. Client reports success to player
        - 1.7.5. Client displays updated board state

DM | Domain Model



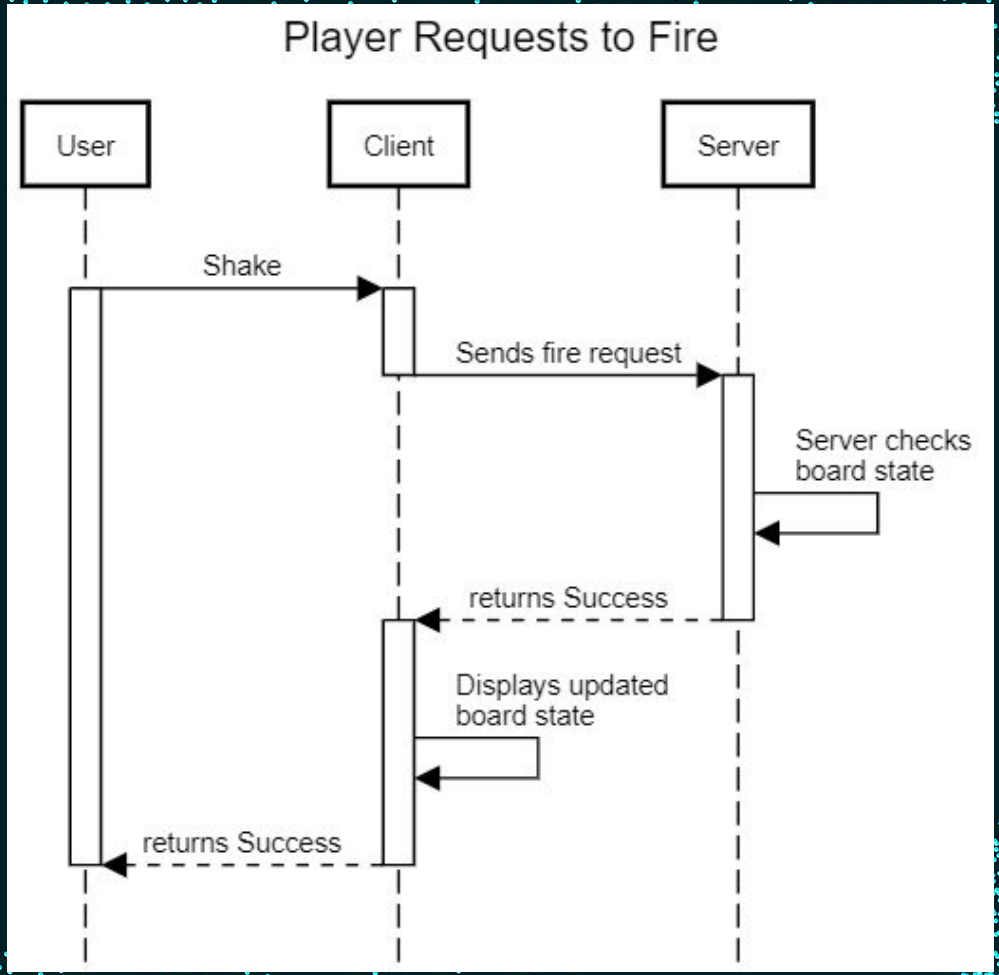
UML Sequence Diagrams

Main Success

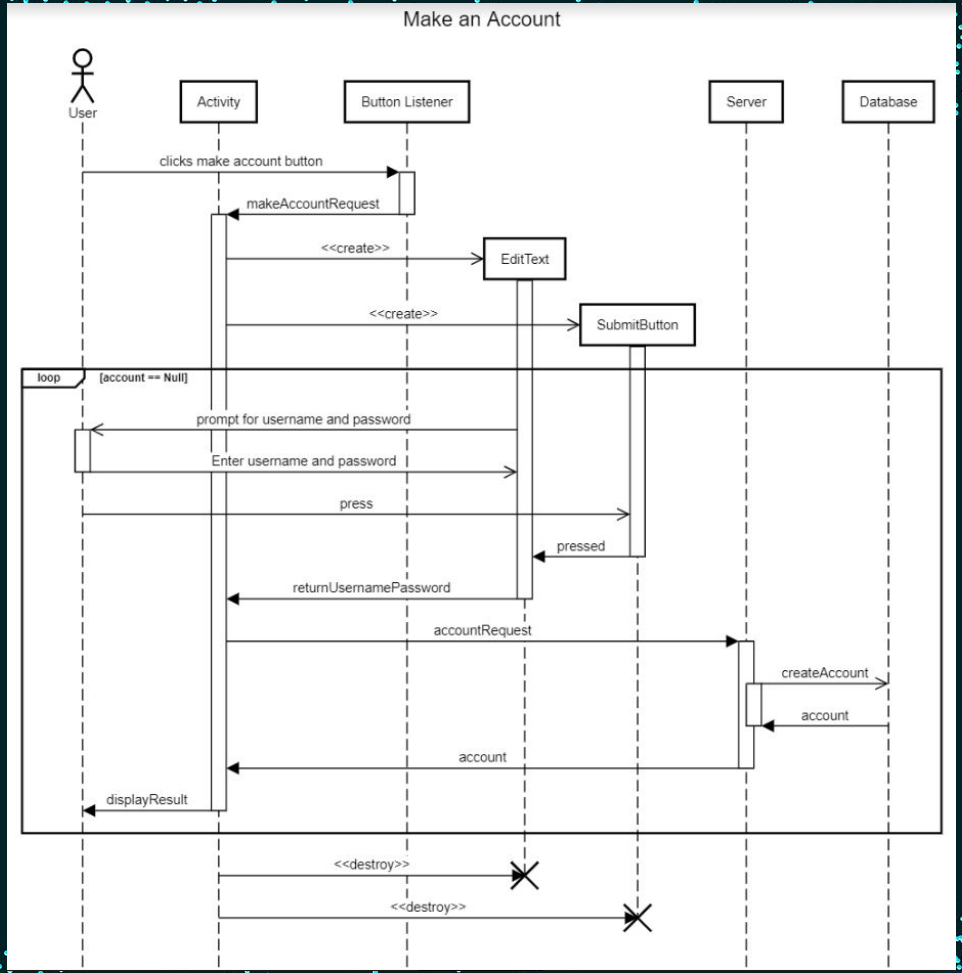
Non-Gameplay

Movement

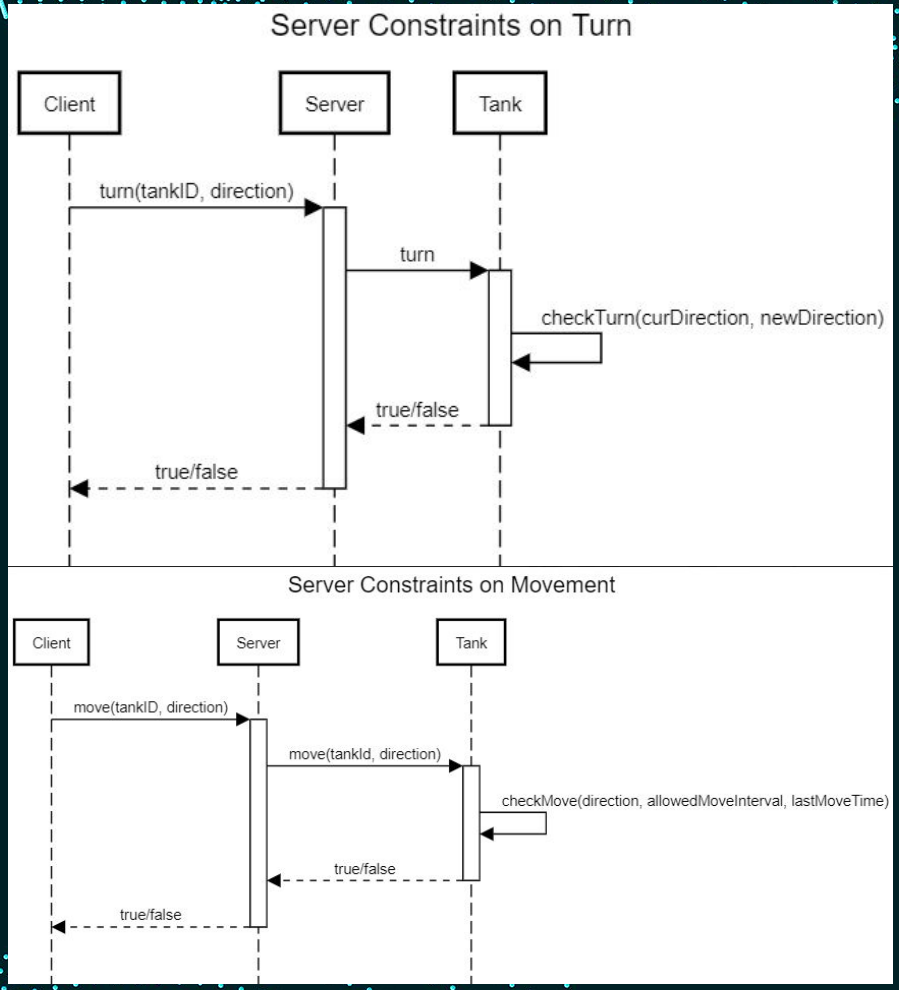
Main Success Scenario



Non-Game Related

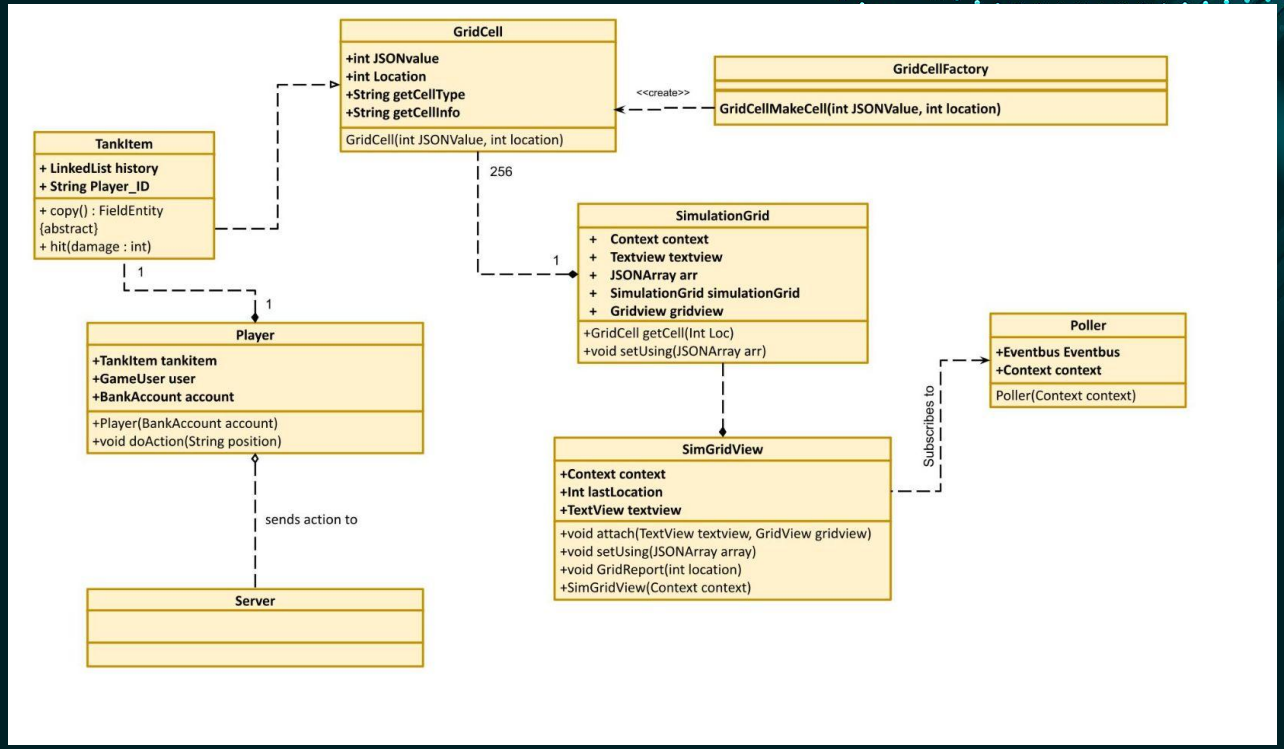


Movement/Turning



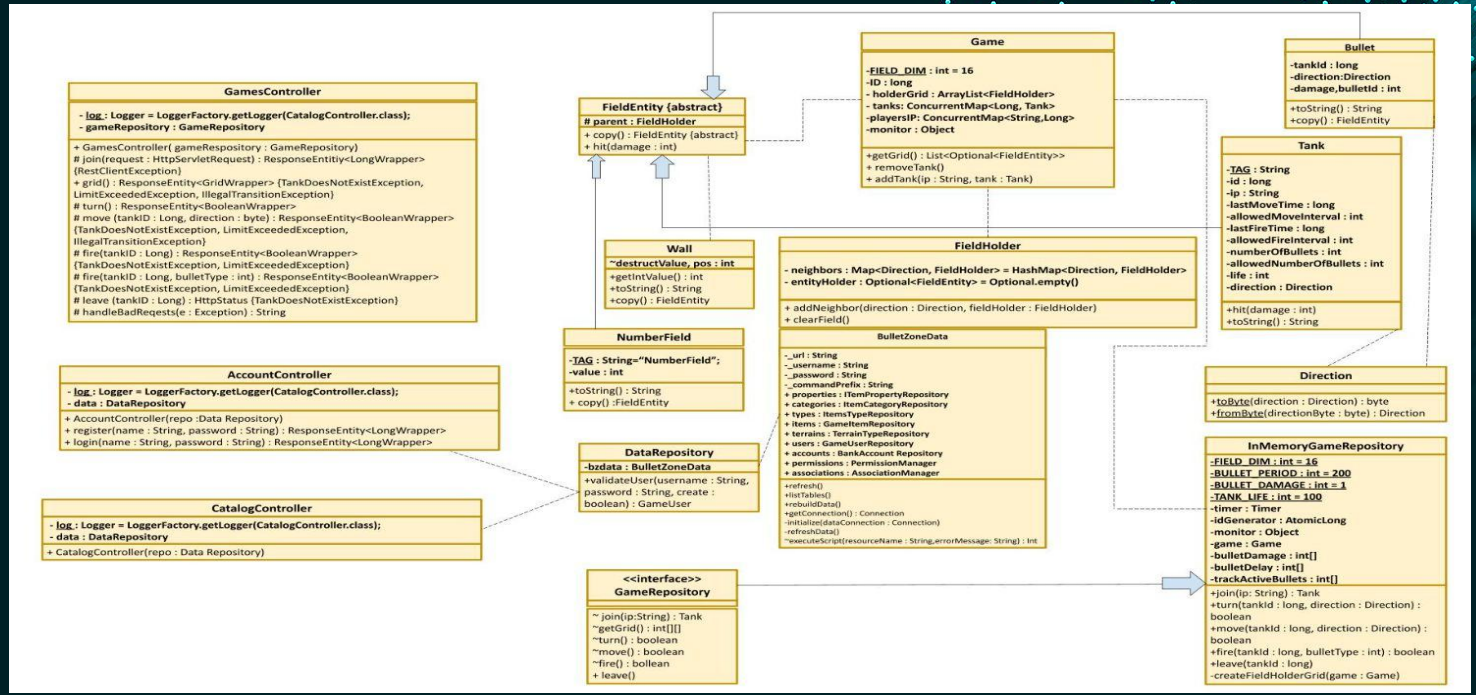
UML Class Diagrams

01 | Client



UML Class Diagrams

02 | Server



Useful Patterns

1. The Command Pattern
    a. The interactions between the database and the server seem to already be through the command pattern.
2. The Simple Factory
    a. Use a factory to determine what kind of entity is made from a number or value from the database
3. Facade Pattern
    a. Used to handle interactions between Android studio activity and client board state
4. Builder
    a. Use a builder to build the game board
5. Observer
    a. Communications from the server to the clients
    b. Clients are subscribers and server is the publisher
    c. Server doesn’t care how many clients they’re are
6. Singleton
    a. Clients use singleton Event Buses to generate an Observer pattern
    b. Singleton Pollers to aid in above pattern

Useful Patterns cont.

Builder

Builder patterns will be useful for setting up the game board. The classes that will participate in this would be:

- Game: this will be on the receiving end of the builder pattern and update with all of the tanks and grid
- Tank: this class will be used to create each tank that is present on the game board.

Observer

Observer Pattern will be useful for monitoring the state change over the course of the game. The classes that will participate in this would be:

- Game: this will update with regards to changes in the observer pattern to reflect a change in game state.
- Tank: this will update with regards to changes in the observer pattern, to reflect the health of a tank or if one gets removed from the game.
- BulletZoneRestClient: this is where the observer pattern gets the state changes from, will update with respect to events that happen in BulletZoneRestClient.

Singleton

Singleton Pattern will be useful for having the get request update at a set interval without issues irregular update intervals.

- GridUpdateEvent: class to hold update events for the event bus.
- GridPollerTask: Sends a new gridupdate to the event bus every 100ms.

THANKS!

Do you have any questions?
matthew.plumlee@unh.edu