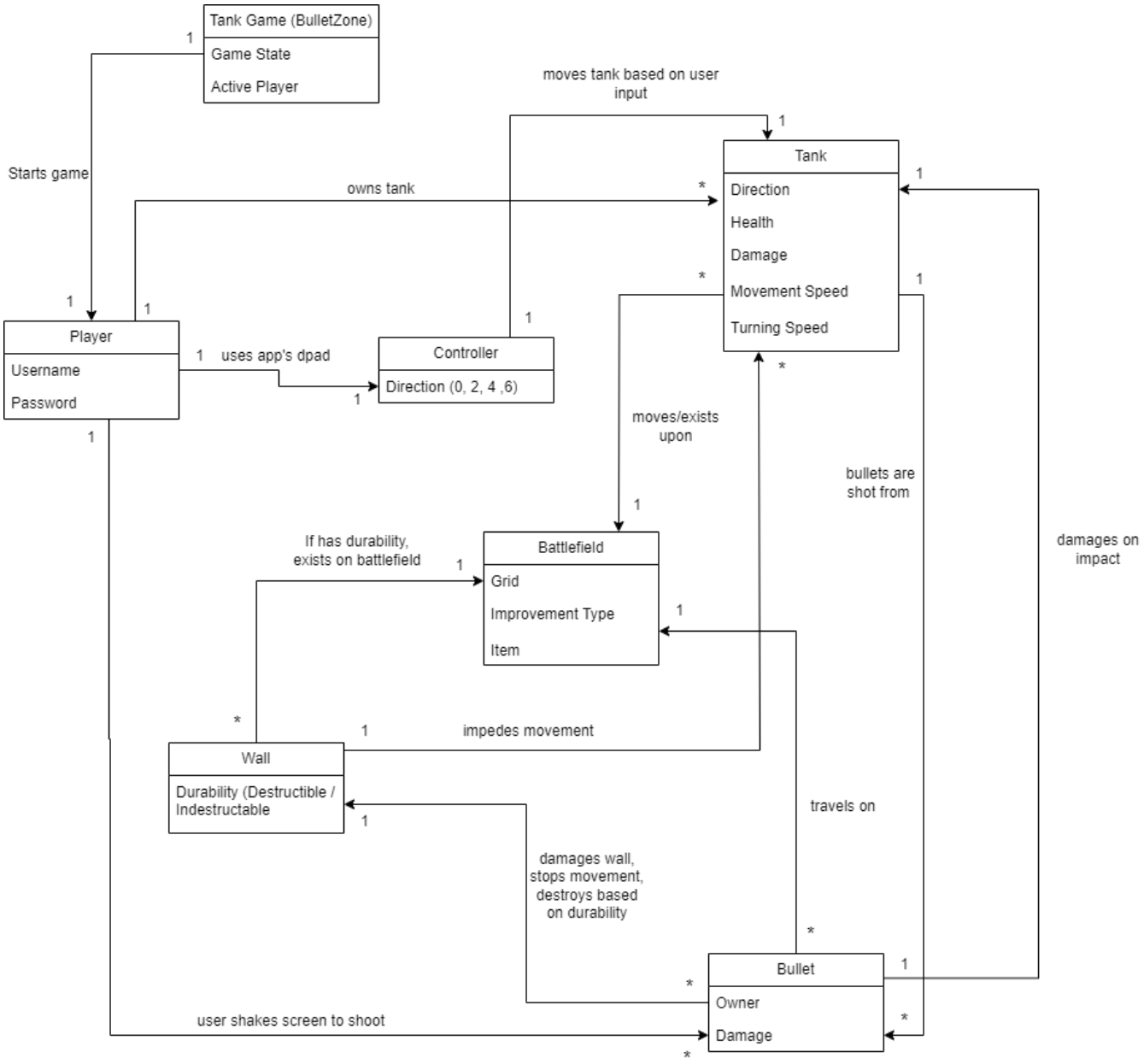
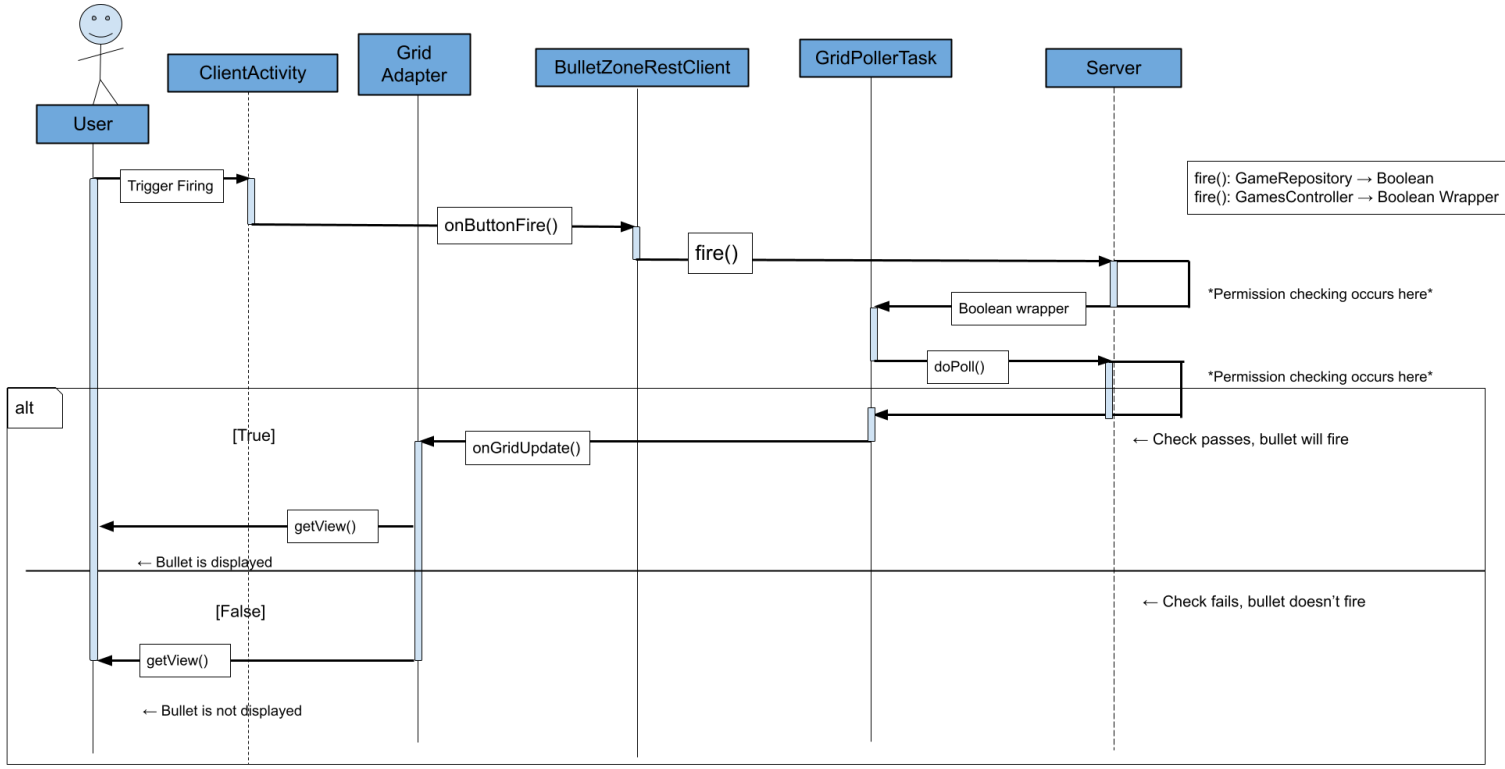
Team Rhea: Seth, Jilly, Jack, Emily, Dart

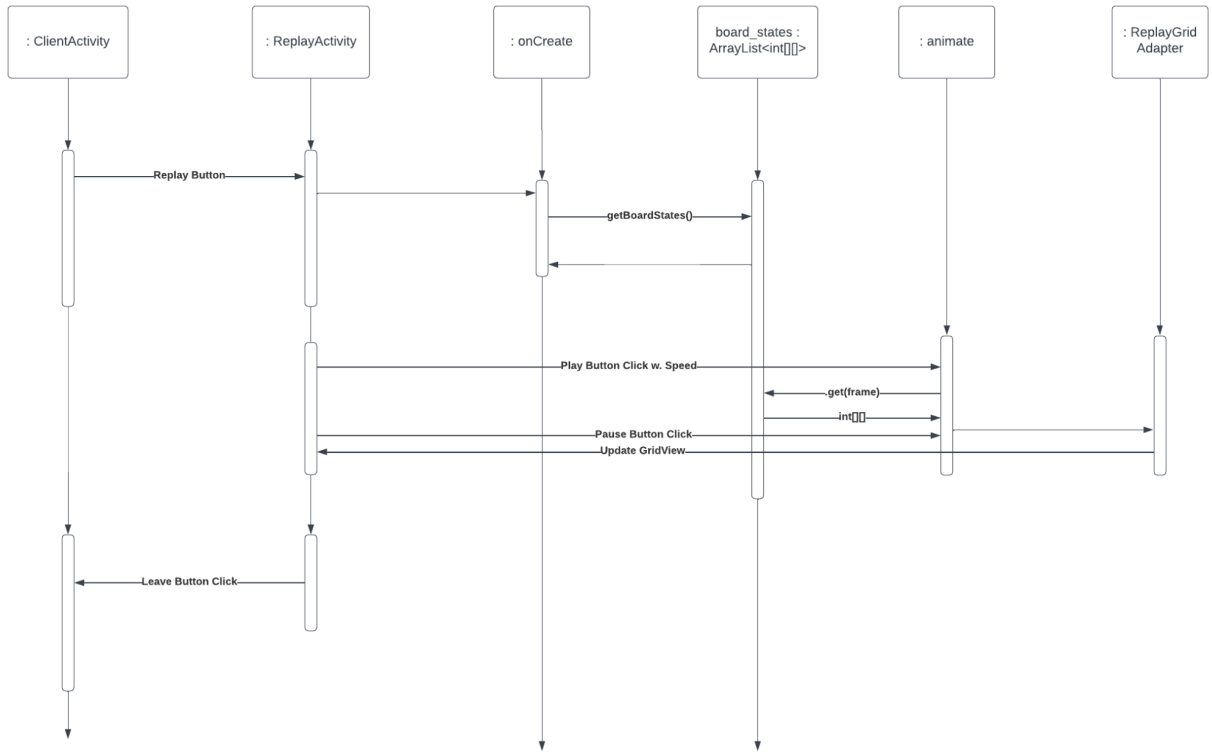
Milestone 0

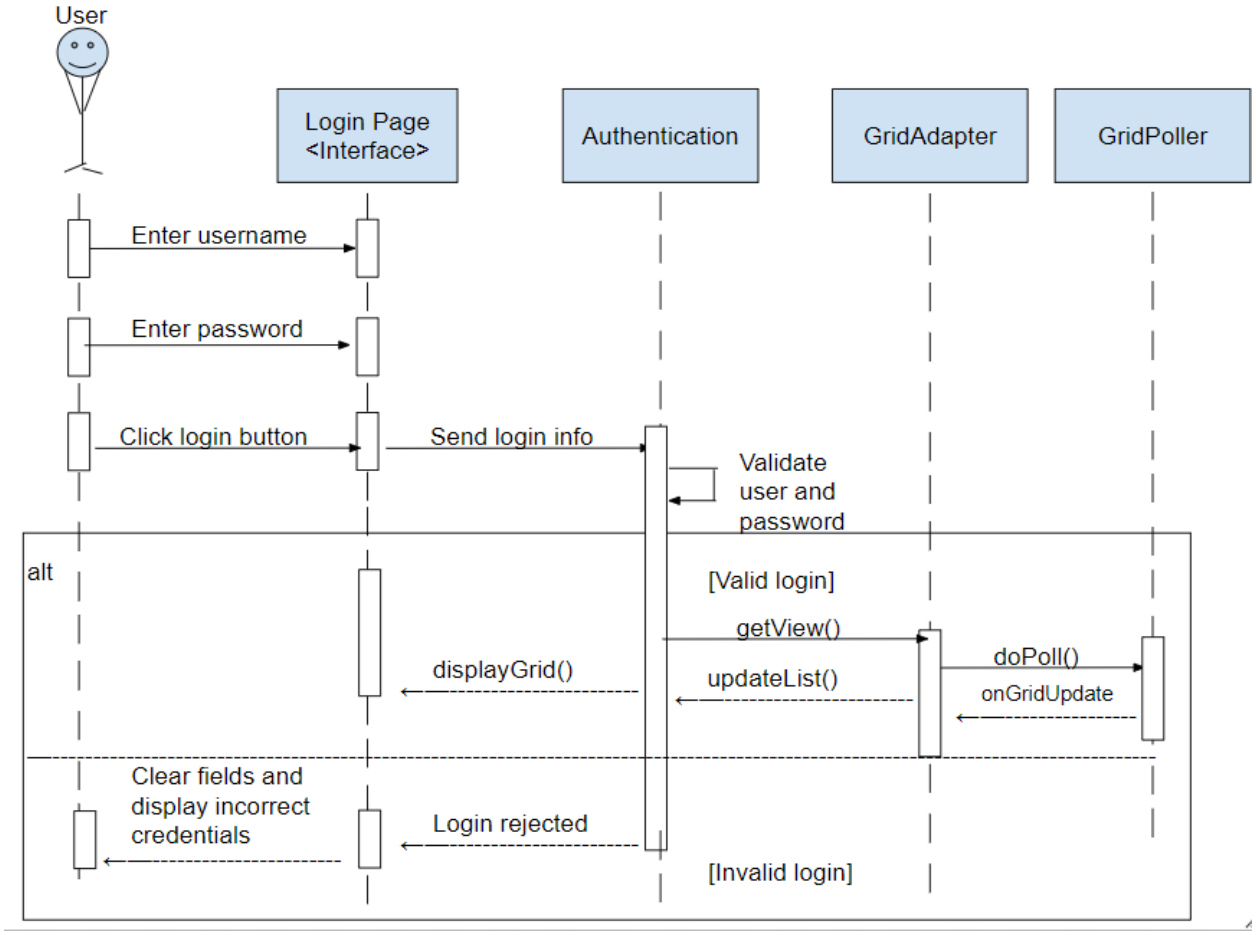
1. **[4 pts]** Domain model (diagram of domain concepts you have identified so far, and their relationships).



2. **[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 "main success scenario" you gave for part 3 listed above. One of the diagrams should involve game-play (controlling the tank) and the other should not. Both diagrams should focus primarily on interactions between software objects within the client.

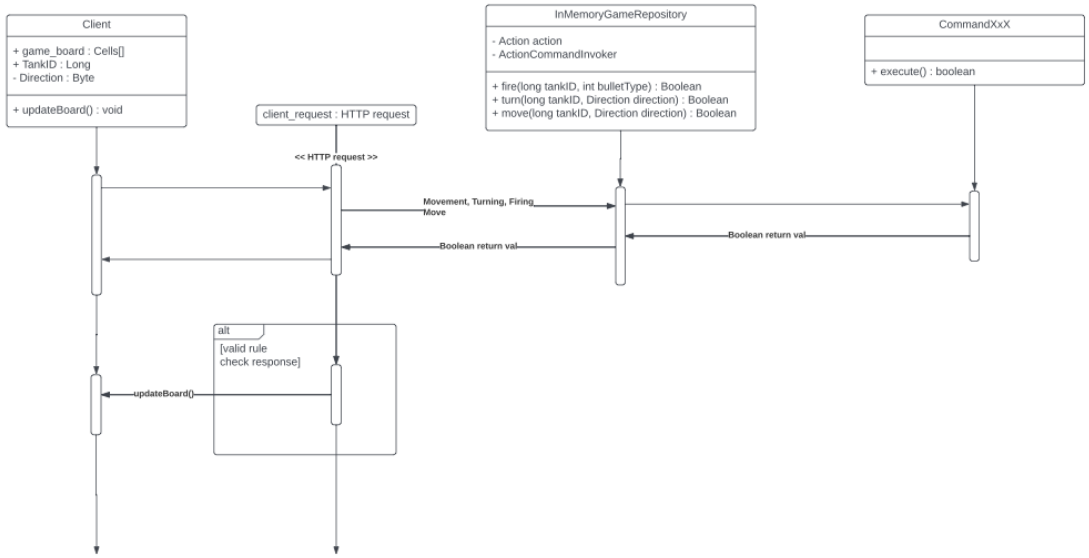

User
ClientActivity
Grid Adapter
BulletZoneRestClient
GridPollerTask
Server
fire(): GameRepository → Boolean
fire(): GamesController → Boolean Wrapper
Trigger Firing
onButtonFire()
fire()
Permission checking occurs here
Boolean wrapper
doPoll()
Permission checking occurs here
alt
[True]
onGridUpdate()
← Check passes, bullet will fire
getView()
← Bullet is displayed
[False]
← Check fails, bullet doesn't fire
getView()
← Bullet is not displayed


: ClientActivity
: ReplayActivity
: onCreate
board_states : ArrayList<int[][]>
: animate
: ReplayGrid Adapter
Replay Button
getBoardStates()
Play Button Click w. Speed
get(frame)
int[][]
Pause Button Click
Update GridView
Leave Button Click

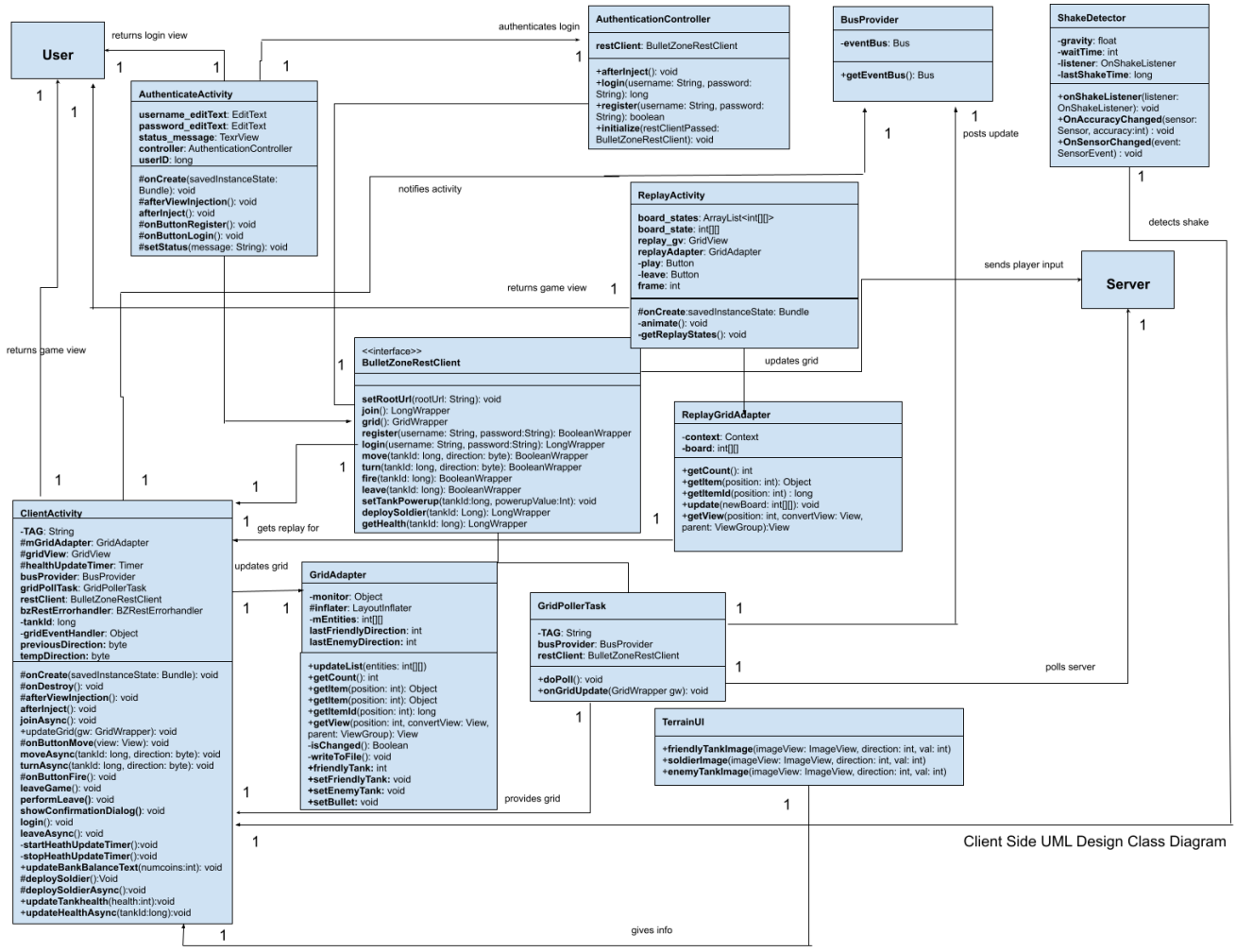


*

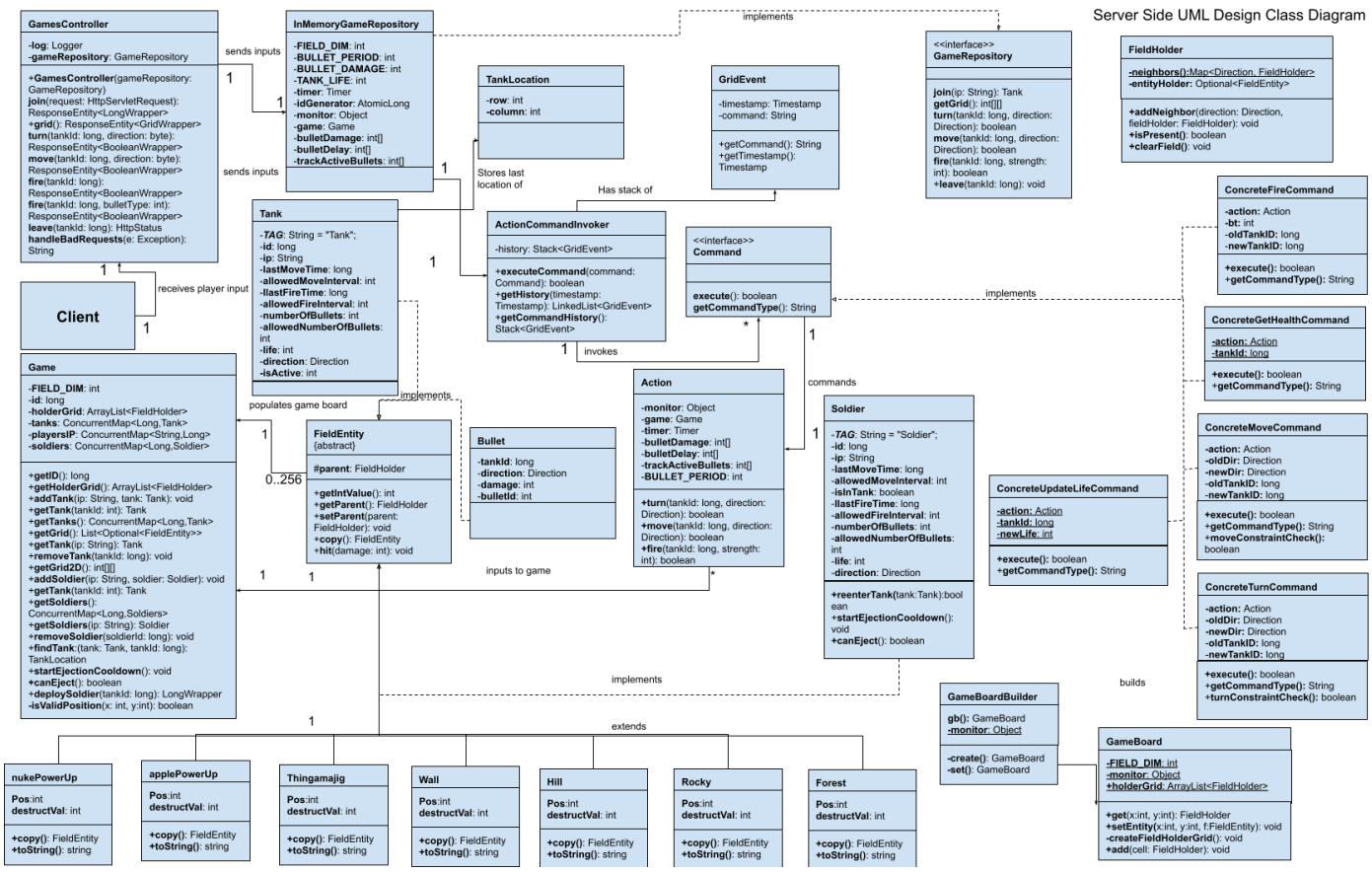
3. **[2 pts]** (At least) one UML sequence diagram illustrating how constraints on turning or movement will be enforced on the server, resulting in a different return value in server responses to turning or movement requests from a client.



4. **[8 pts]** UML design class diagram(s) that supports joining a game, plus the gameplay use-cases... preferably should address the needs of the entire system as you know them so far. It usually works best to have one diagram for the client and one for the server.



Client Side UML Design Class Diagram



5. **[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.

In the class ClientActivity, multiple patterns can be found. The singleton pattern is found in the usage of BusProvider, where it will provide a single instance of an event bus to its client, GridPoller, which uses. Its class operation is the ClientActivity. Another pattern involving ClientActivity is the facade pattern. The clients of this pattern would be the ClientActivity, and BusProvider.The facade would be the Poller class, and the subsystem classes would be the classes contained in the server.

Then there is usage of a MVC (model-view-controller) pattern through classes BulletZoneRestClient, GridView and ClientActivity. BulletZoneRestClient is going to serve as our Model, doing all processes surrounding data logic (login(), turn(), fire()) and is also frequently called upon by ClientActivity, or the brain/controller of the program. GridView is the view UI component (View) responsible for

displaying the grid to the user. ClientActivity will handle as our controller, tackling human interactions within the app and updating the view based on those actions.