Milestone 0: Design Checkpoint
Group A: Cooper Montgomery, Chris Nippert, Ryan Pereira, Aidan Small, Richa Yadav

Functional Requirements

Client:

Account Management:

- User should be able to login to an account
- User should be able to register an account
- User should be able to manage their tokens
- User should be able to manage their account

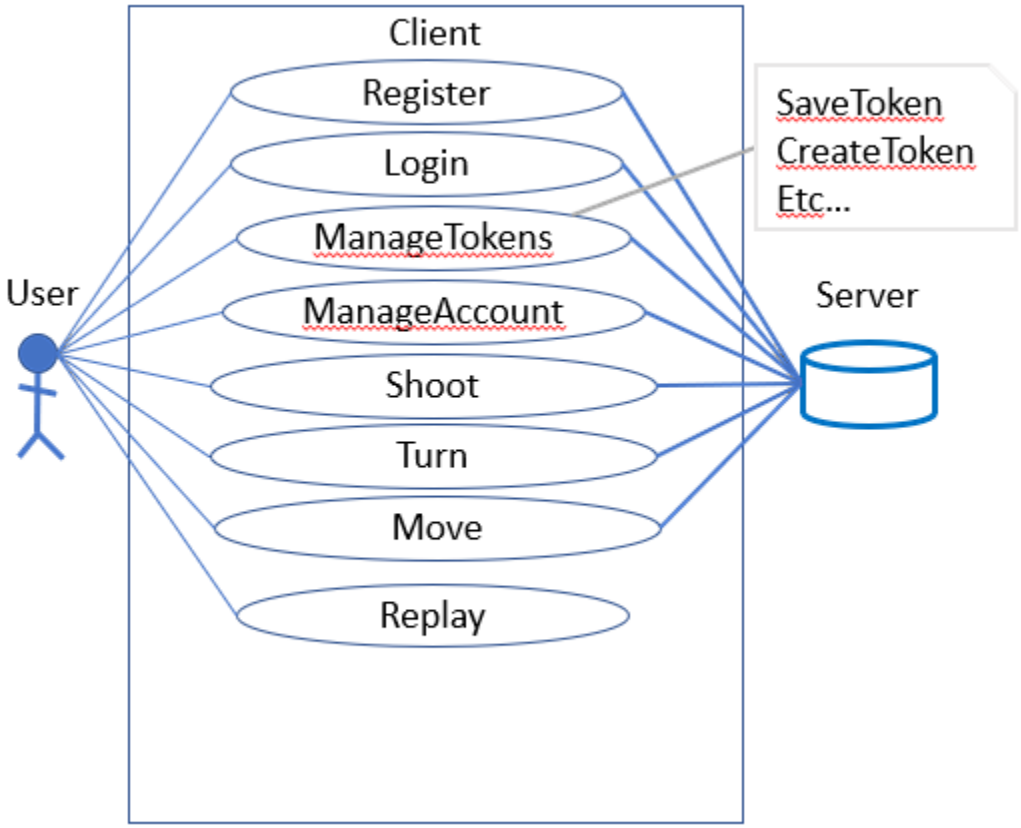
Gameplay:

- Client should be able to shoot, turn, and move using the server to validate these movements
- Client should be able to store game history for the replay functionality
- Client should be able to replay games using the game history
- Client should be able to start the poller to get updates from the server

Server:

- Server should be able to initialize the game board as well as the size of the board
- Server should manage the game board and game time
- Server should be responsible for keeping record of player account data
- Server should be able to synchronize the clients to keep them at the same point in the game so the moves seen are the most recent for both
- Server should be able to to update clients synchronously allowing for multiple clients to play the game together at once
- Server should be able to interact with database to get account information as well as game board information
- Server should be able to store event history to send back to client to update game board display
- Server should be able to determine if a move is valid
- Server should be able to determine if a turn is valid
- Server should be able to determine if a tank is allowed to shoot

Use-Cases



- Register
- Login
- ManageTokens
- ManageAccount
- Shoot
- Turn
- Move
- Replay

Main Success Scenarios

Register:

1. The user is prompted with the register page
2. The user inputs information into text fields
3. The client verifies the info is legal and will not harm the server or database
4. If Inputs are valid
5. Then the client sends the info to the server to check if the user already exists
6. If user does not exist
7. Then the server stores account information into database
8. The server then logs in the user and brings them to the game screen

Extensions:

5a.

5a.1 The client prompts the user that their credentials are invalid/illegal (illegal characters, et cetera)

5a.2 Returns the user to the register page to retry credentials (step 1)

7a.

7a.1. Server sends an error message back to the user saying “User Already Exists. Login?”

7a.2 Returns the user to login page to attempt to login (step1)

Login:

1. The user is prompted with with the login page
2. User inputs information into text fields
3. The client validates the login info is legal to protect the database/server
4. If the inputs are valid
5. Then the info is sent to the server
6. If the server verifies the specified user account exists
7. Then the user is logged into the account
8. The game screen is displayed to the user ready for interaction

Extensions:

5a.

5a.1 The client prompts the user that their credentials are invalid (illegal characters, et cetera)

5a.2 Returns the user to the login page requesting new informations (step 1)

7a.

7a.1. Server sends an error message back to the user saying “User does not exist. Register?”

7a.2. Returns the user to the login page requesting new informations (step 1)

Move:

1. The user presses a key to move a token
2. The client processes which key was pressed
3. The move command is sent to the server
4. The server validates the move (determines if the user can move based on the last time they moved)
5. If the move is valid
6. Then the server updates the current game state
7. The client updates all user's UI with the new game state
8. The next user makes a move if wanted

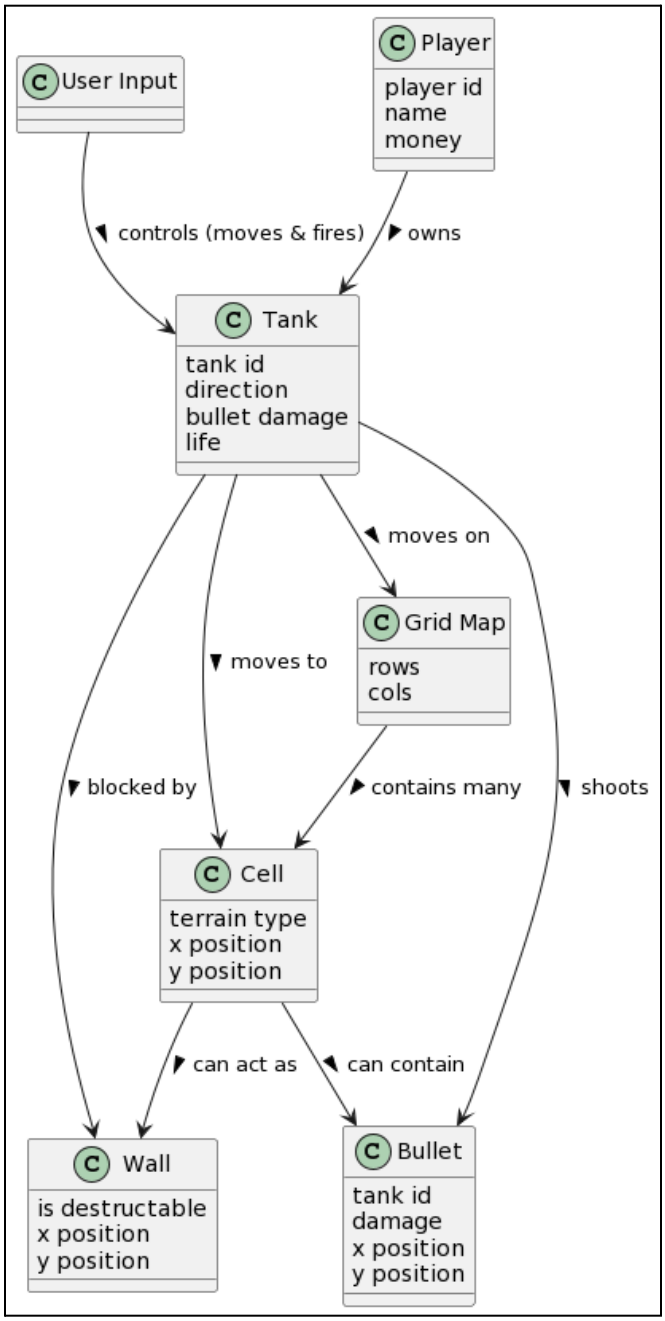
Extensions:

6a.

6a.1. The move is not valid, so the move does not alter the server's game state

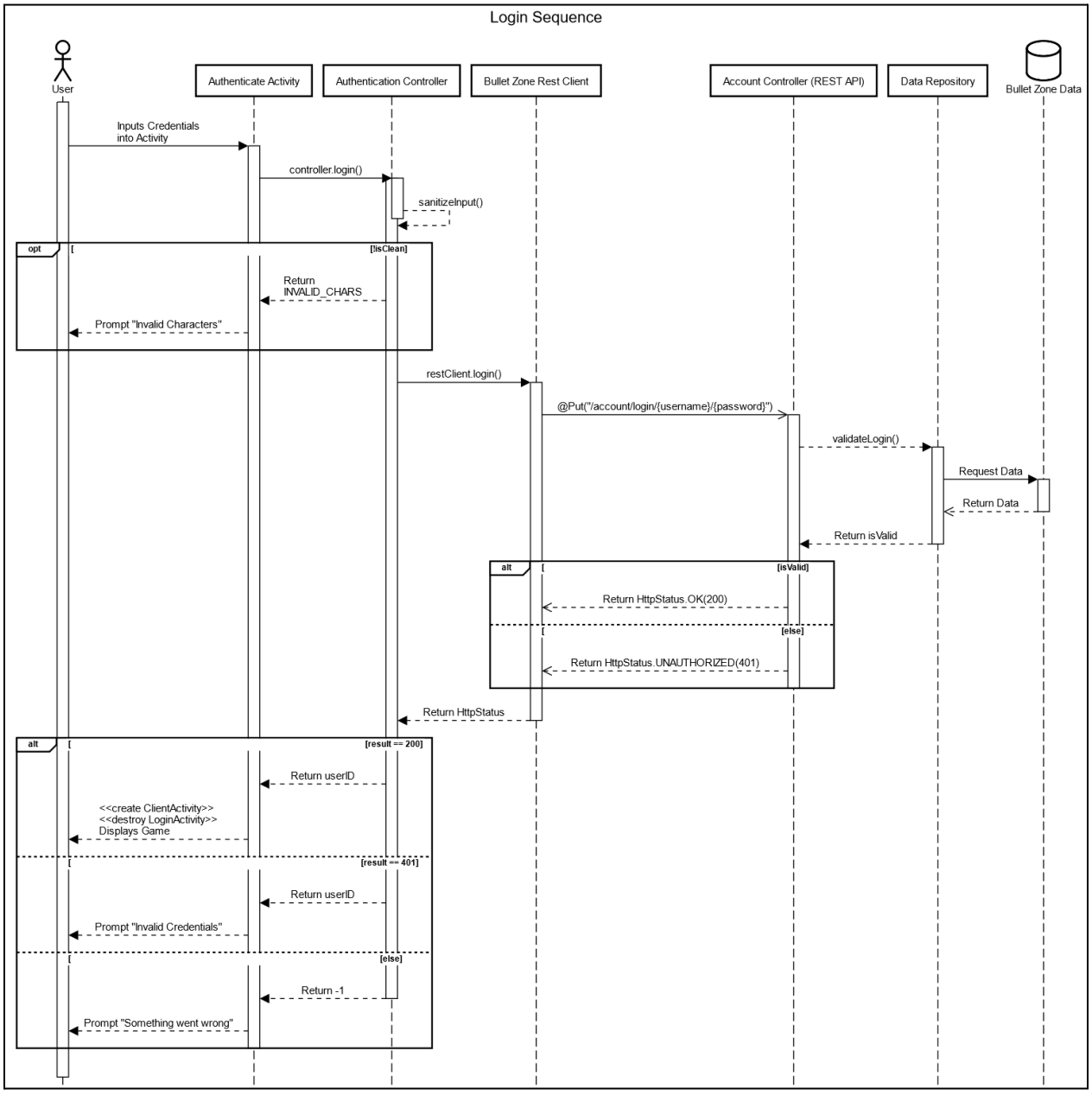
6a.2. Client and server do nothing and waits for next player to make a move (step 1)

Domain Model*

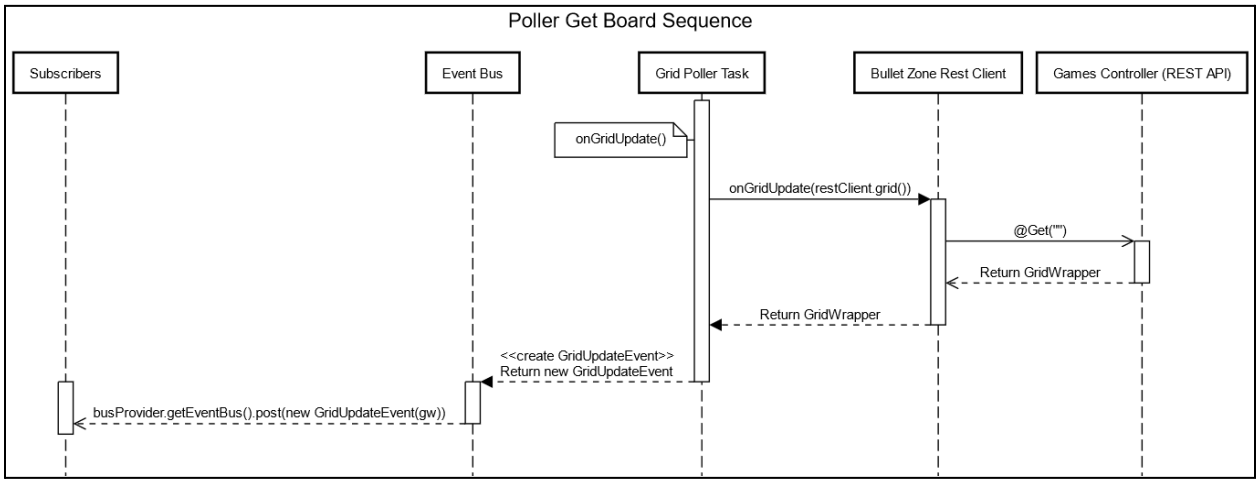


UML Sequence Diagrams*

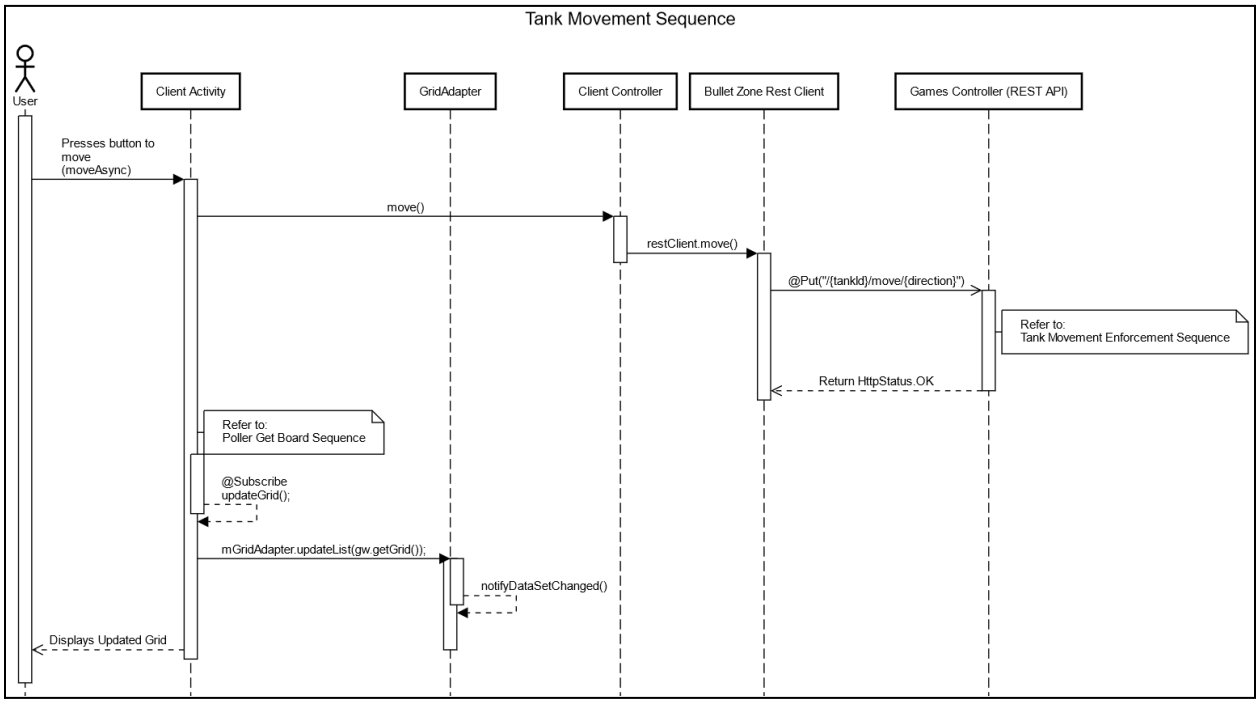
Login



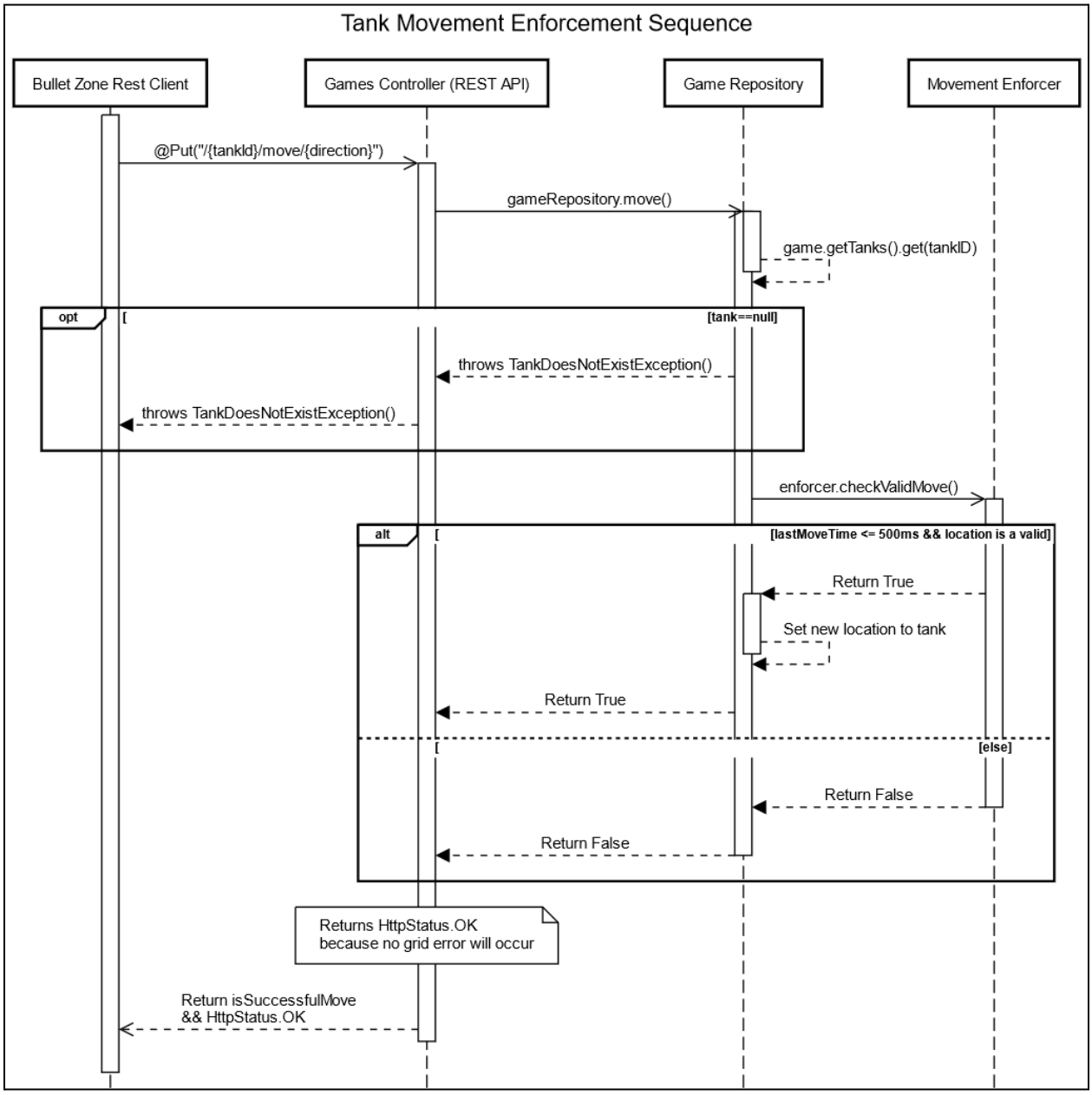
Poller Get Board



Tank Movement

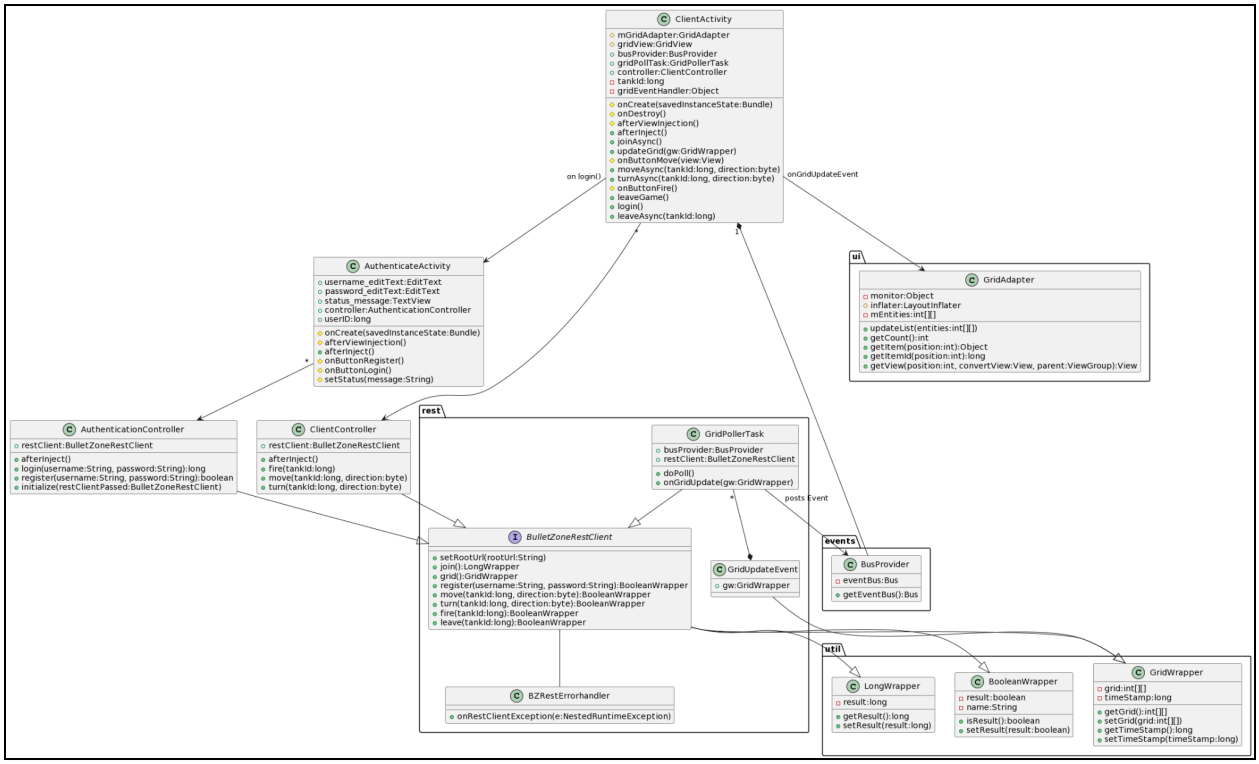


Tank Movement Enforcement

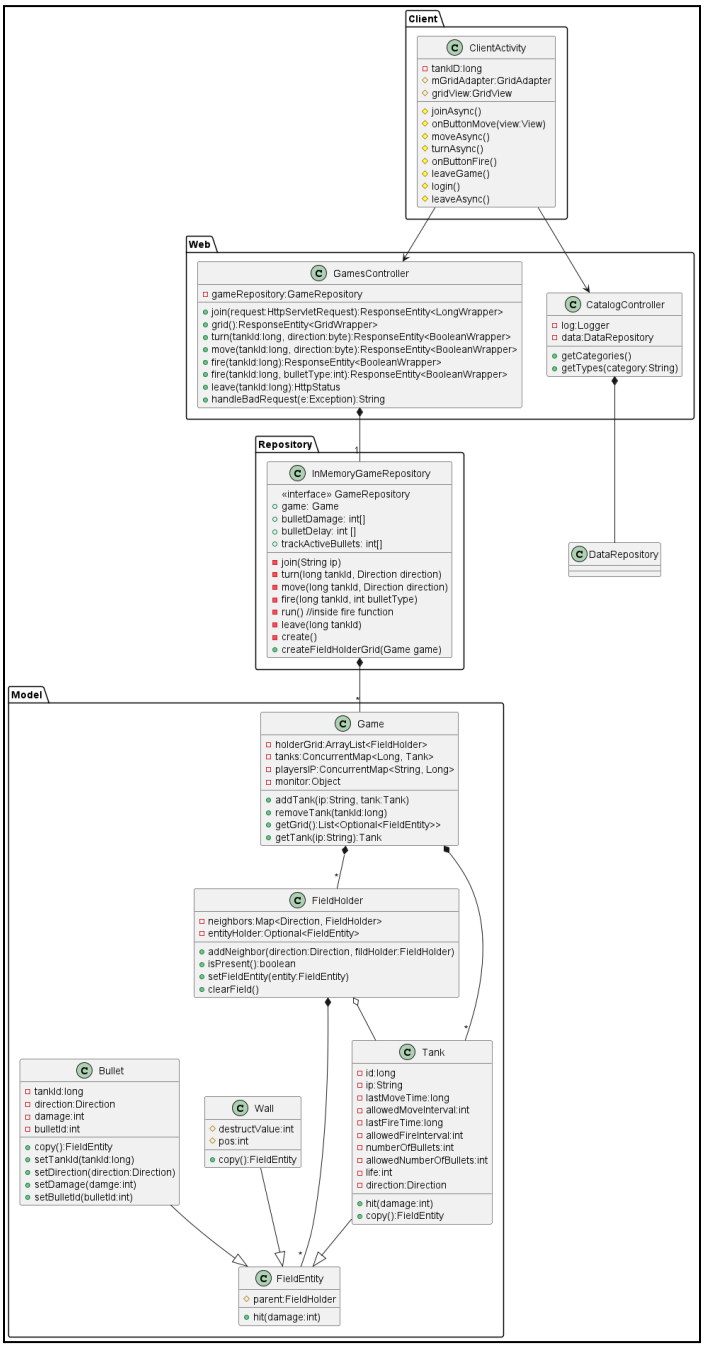


UML Design Class Diagrams*

Client



Server



**Nicer diagrams on the gitlab*

Patterns

Singleton:

- BusProvider (revised 2)
  BusProvider is an example of a singleton class since it includes the required variable, functions, and private constructor. It is used throughout the entire client side of the app. In the client it is used in the GridPollerTask to update the grid and the ClientActivity on deletion. This would best fit into the controller part of the MVC model since it sends a signal to the subscribers when an action is complete (controls them).

Observer:

- GridPollerTask (revised 2)
  The poller class, GridPollerTask contains the function that we will use to be constantly updating our board. This is the observer itself since it is updating when something changes on the server. In the MVC diagram this would be an example of a controller. This is because it is sending information to be processed by the model later on.
- BusProvider (revised 2)
  While BusProvider is a Singleton, it is also a member of the Observer pattern, where it is the publisher for the data grabbed by GridPollerTask. This is an example of the concrete observer, because it is directly communicating with the subject. This is another example of a controller because it is controlling the signal being sent to something else.
- GridUpdateEvent (revised 2)
  This is the concrete subject because it is setting the state of the subject in its constructor. It also contains information on the state of the subject. This is an aspect of the model part of the MVC model since it is manipulating the visuals the user will see.
- GridWrapper (revised 2)
  GridUpdateEvent will be subscribed to the publisher, so that it can get the data each time it is updated (where it then sends to GridWrapper, to update the item. This is the subject since it is used to attach the concrete subject to the observers. This is another aspect of something within the model because it is used to hold information for the view.

Facade:

- GameItemContainer (revised 2)
  This class consists mostly of getters to get information about different items in the application. This could be seen as it getting info from different subsystems in the application. There is also the ability to manipulate these subsystems such as remove

and add items. This is an aspect of the controller part of the MVC diagram. It is used to get information and change things about the model.

Simple Factory:

- Direction (revised 2)
  The direction class is a simple factory because there is a switch inside of the class that determines which direction is being selected. It sets the direction using an enum's value and also has the ability to convert this to a corresponding integer. This is part of the model because it is used to update the view for the user. It directly changes elements in the grid view.

Builder:

- GameBoardBuilder (revised 2)
  This is a theoretical class that is the builder itself in the builder pattern since it will be responsible for building the parts of the product. This would be part of the model because it is called to do the first update of the game board. This board is displayed to users as a view.
- Game (revised 2)
  This is the director because it is responsible for using the builder to create the products. This holds the product within it to be manipulated by the builder. This would be part of the model because it is responsible for updating multiple things in the view directly and indirectly.
- GameBoard (revised 2)
  This is the product because it is being held by Game and is being created by GameBoardBuilder. It has all the information for the stuff that will make up the board. The would best fit in the view part of the MVC model because it holds the values that are displayed on the UI. Also, it is updated by another class in the model.

Command:

- ClientReplay (revised 2)
  The Client Replay function would use a command pattern to keep track of events so that the replay can play forward from a certain point. This would be the receiver in the command pattern. This would be part of the view since it is responsible for displaying the previous game played.
- ClientPlaybackEngine (revised 2)
  The Client must be able to play events that are sent by the server. This allows the client to keep the board up to date by executing the events that are sent. This would be the concrete command in the pattern. This would likely be part of the model because it will most likely be used to update the replay. Thus, updating the view.

- ServerHistory (revised 2)
  The server must keep track of events using the Command pattern so that it is able to send commands to others. This would be the invoker in the pattern. This would be part of the controller because it would manipulate how the playback engine updates the client replay. It would control how the model updates the view.
- ServerEventCommands (the commands sent to the client) (revised 2)
  The server must be able to send the client a list of commands for it to execute so that the client stays up to date with what is happening. This would be the command itself in the pattern. This would also be part of the controller since it is sending data to the client and this will manipulate how it interacts with the views.