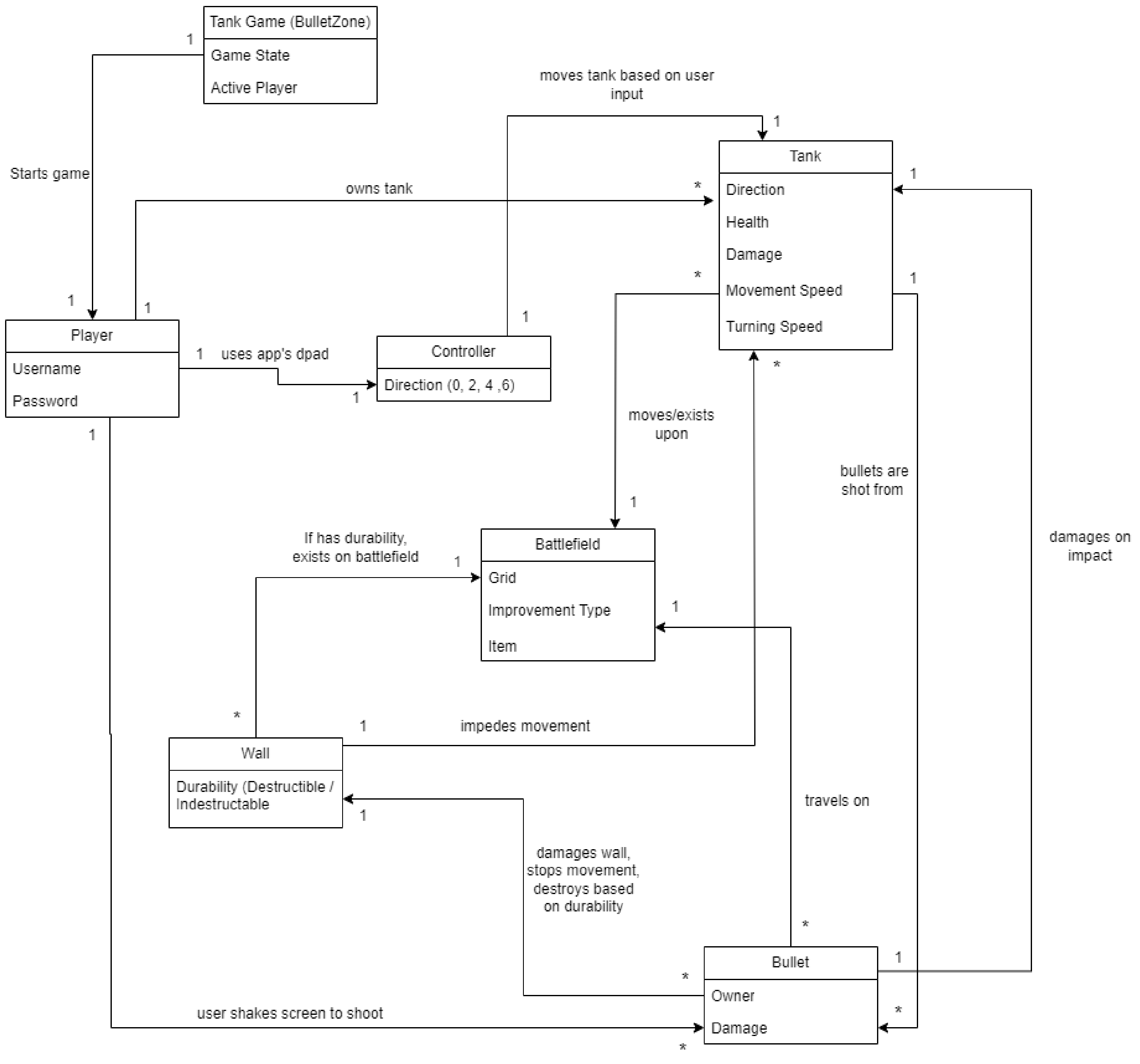
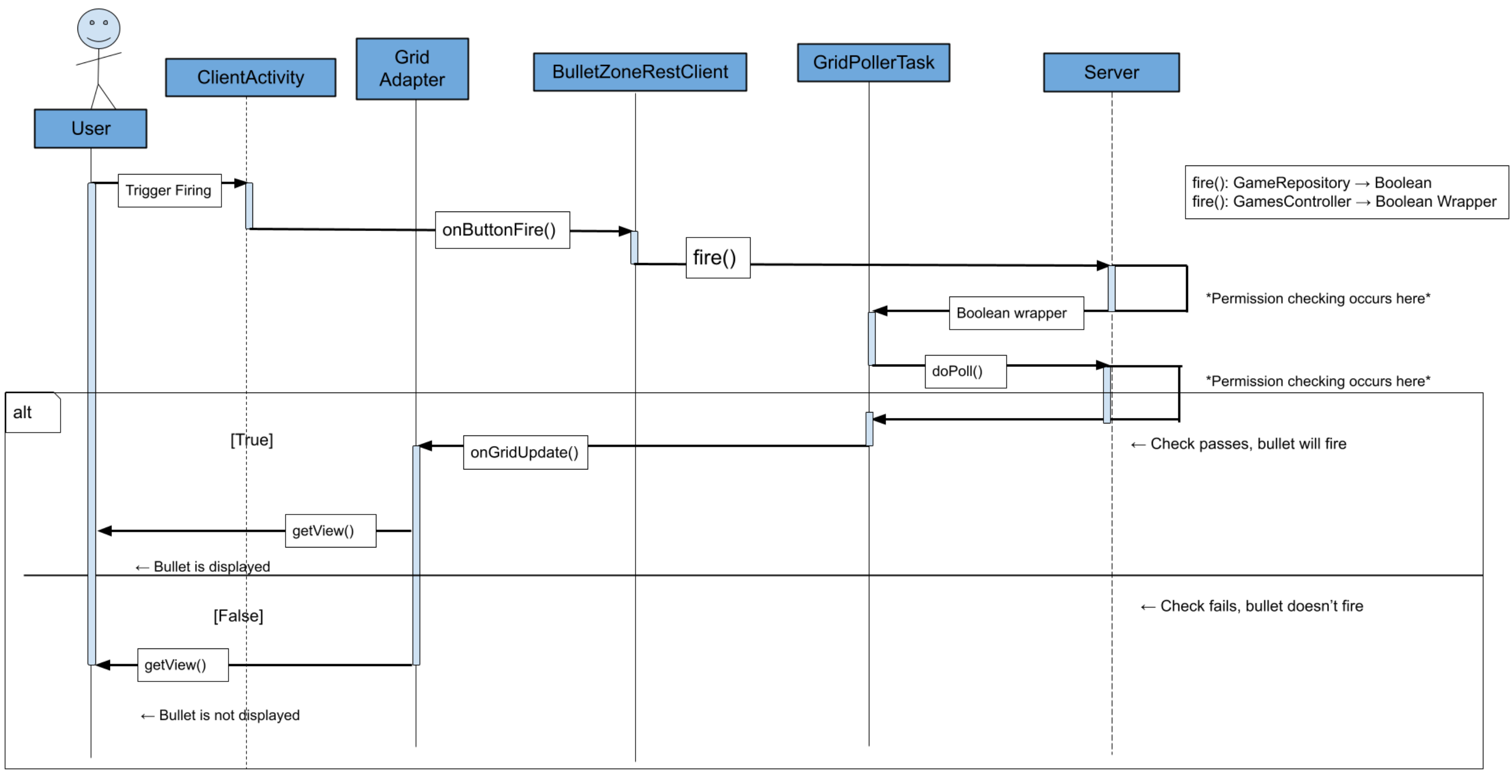
Team Rhea: Seth, Jilly, Jack, Emily, Dart

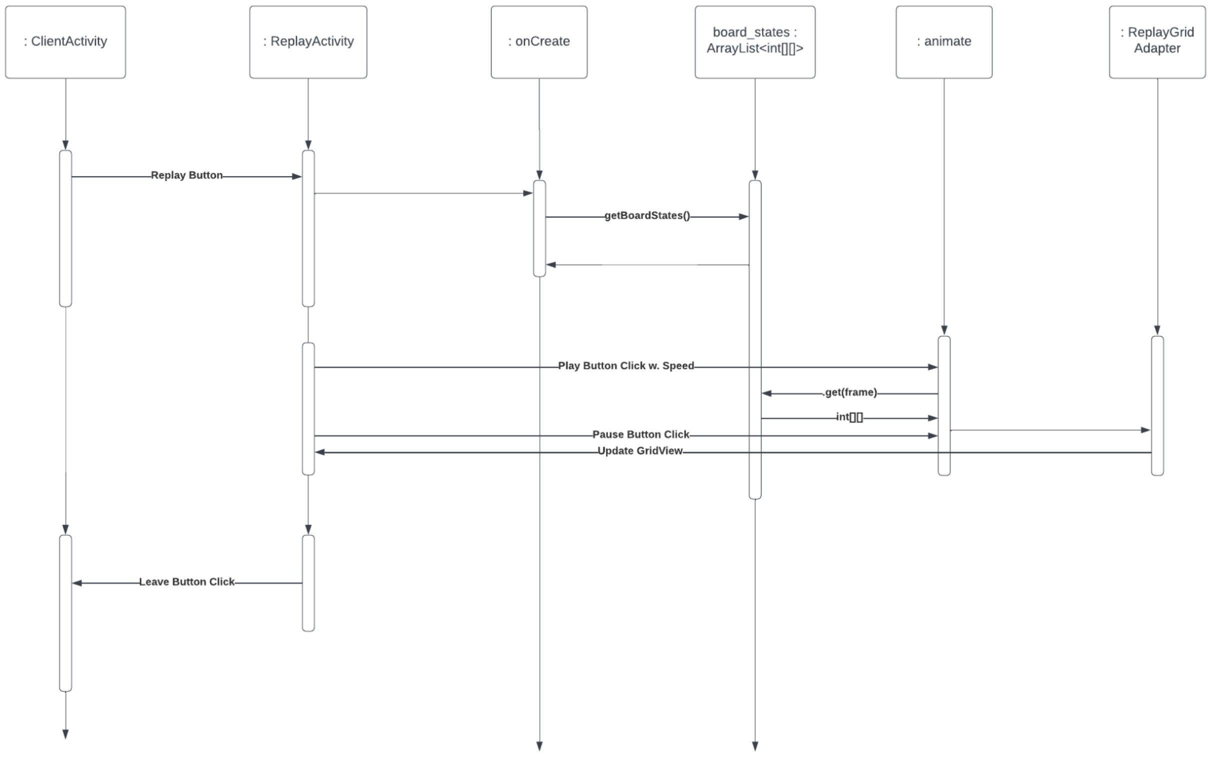
Milestone 0

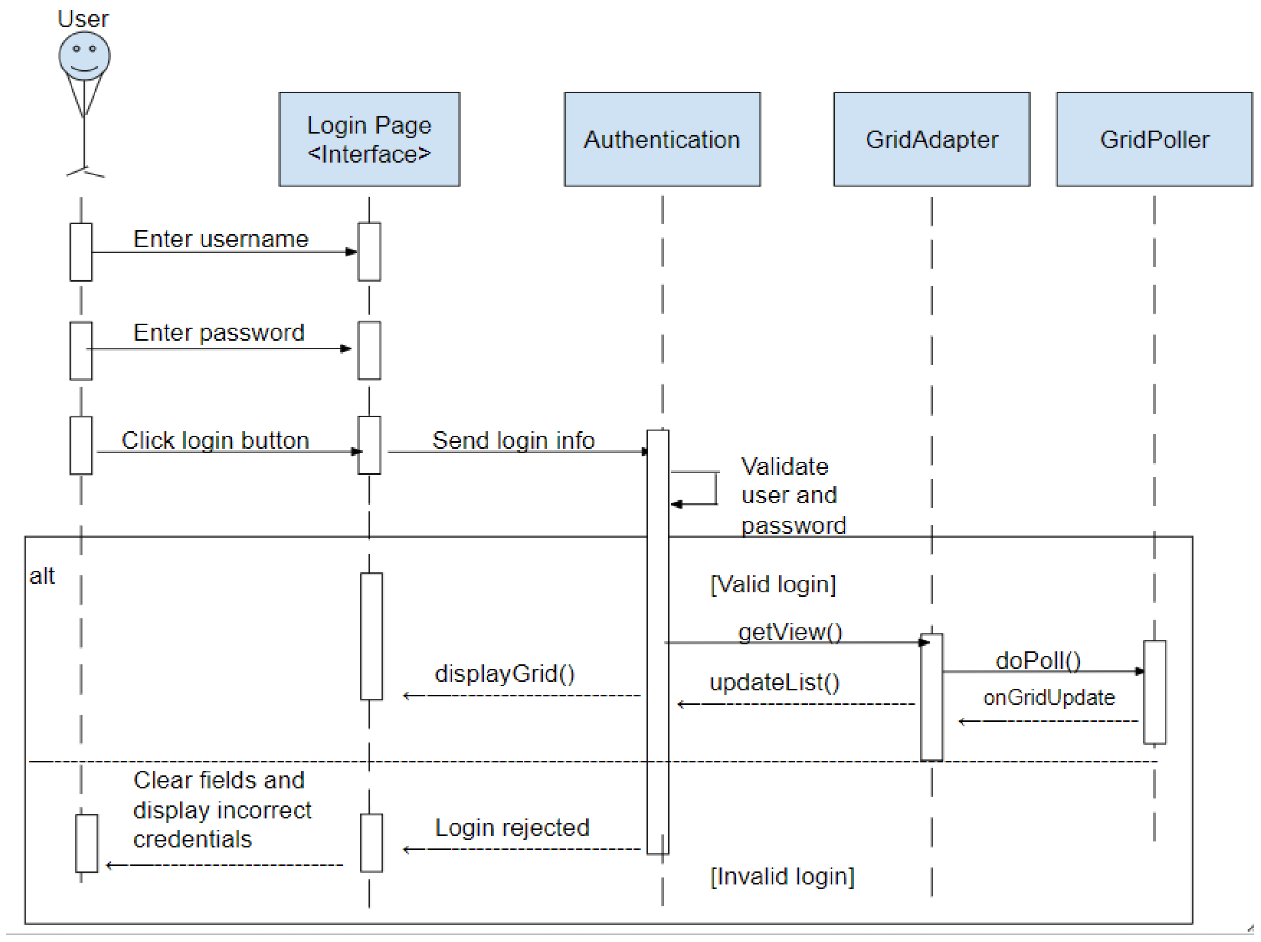
1. **[4 pts]** Domain model (diagram of domain concepts you have identified so far, and their relationships).



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.

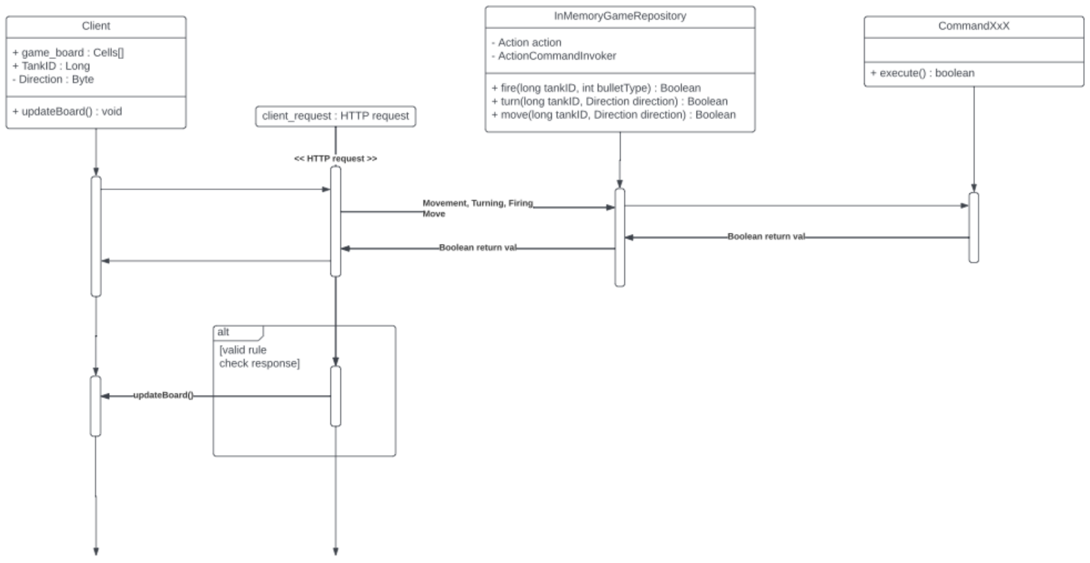

User
ClientActivity
Grid Adapter
BulletZoneRestClient
GridPollerTask
Server
fire(): GameRepository → Boolean
fire(): GamesController → Boolean Wrapper
Trigger Firing
onButtonFire()
fire()
Permission checking occurs here
Boolean wrapper
doPoll()
Permission checking occurs here
alt
[True]
onGridUpdate()
← 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

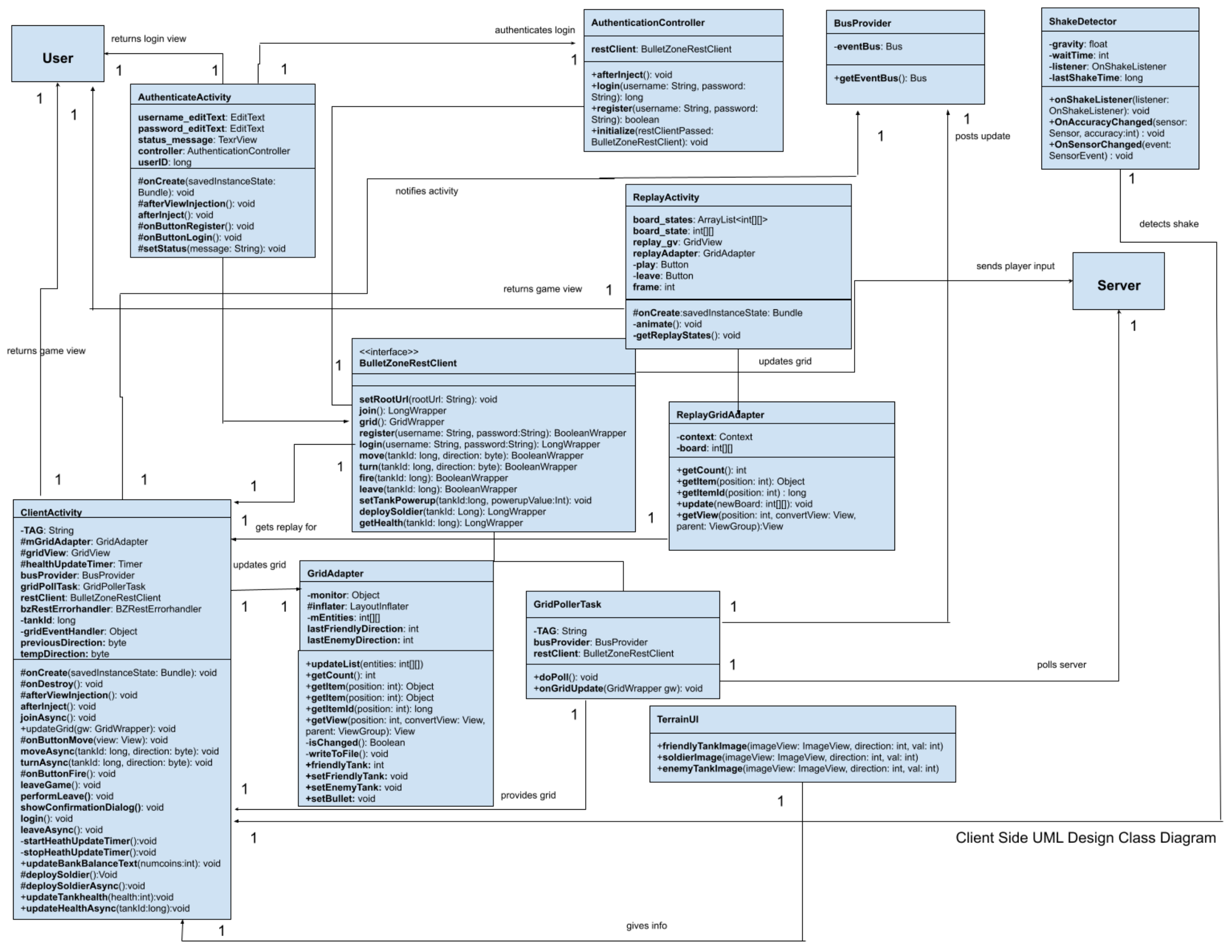


*

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.

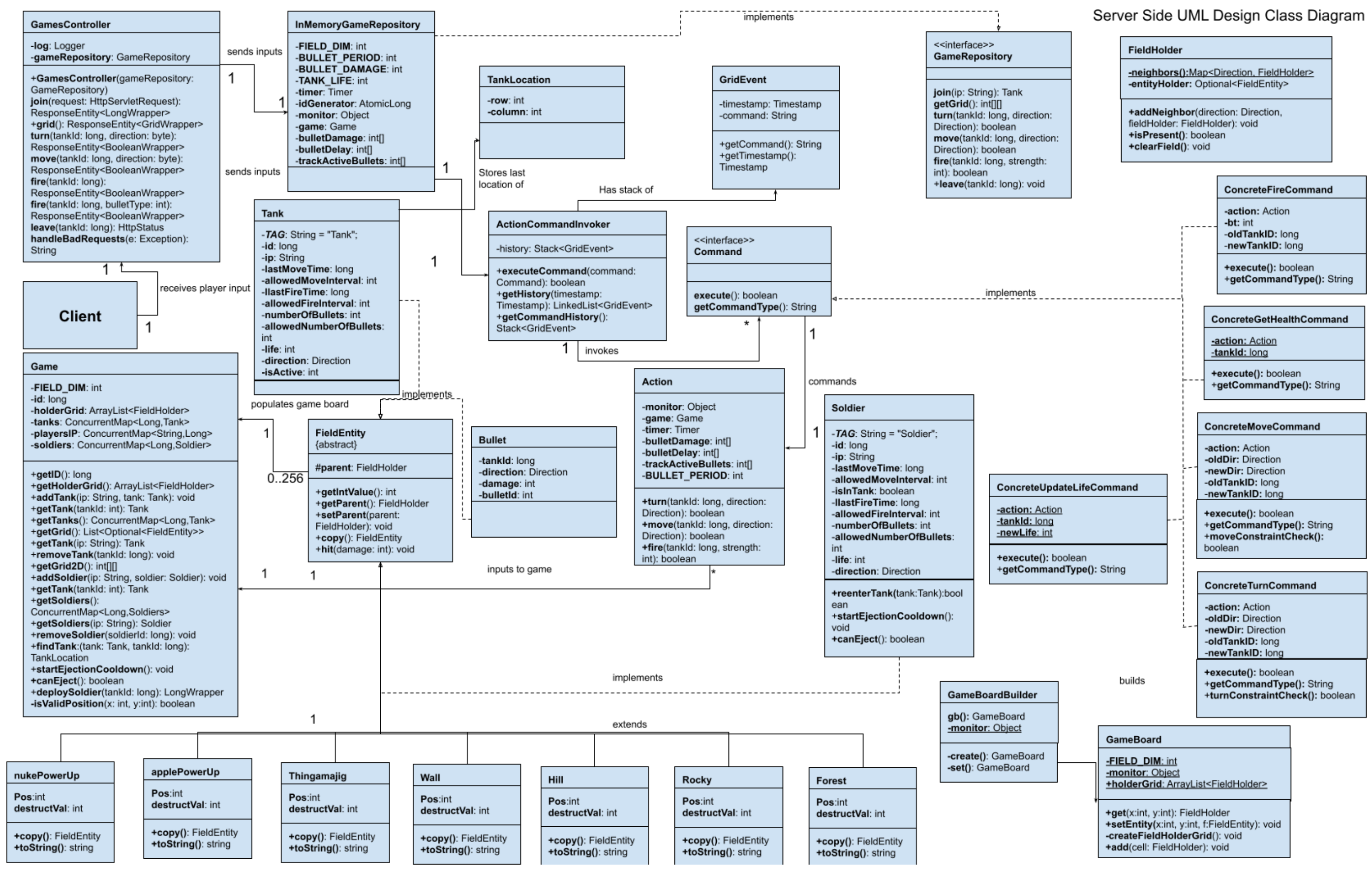


4. **[8 pts]** UML design class diagram(s) that supports joining a game, plus the gameplay use-cases... preferably should address the needs of the entire system as you know them so far. It usually works best to have one diagram for the client and one for the server.


User
returns login view
authenticates login
AuthenticationController
restClient: BulletZoneRestClient
+afterInject(): void
+login(username: String, password: String): long
+register(username: String, password: String): boolean
+initialize(restClientPassed: BulletZoneRestClient): void
BusProvider
-eventBus: Bus
+getEventBus(): Bus
ShakeDetector
-gravity: float
-waitTime: int
-listener: OnShakeListener
-lastShakeTime: long
+onShakeListener(listener: OnShakeListener): void
+OnAccuracyChanged(sensor: Sensor, accuracy:int) : void
+OnSensorChanged(event: SensorEvent) : void
AuthenticateActivity
username_editText: EditText
password_editText: EditText
status_message: TexrView
controller: AuthenticationController
userID: long
#onCreate(savedInstanceState: Bundle): void
#afterViewInjection(): void
afterInject(): void
#onButtonRegister(): void
#onButtonLogin(): void
#setStatus(message: String): void
notifies activity
posts update
detects shake
ReplayActivity
board_states: ArrayList<int[][]>
board_state: int[][]
replay_gv: GridView
replayAdapter: GridAdapter
-play: Button
-leave: Button
frame: int
#onCreate:savedInstanceState: Bundle
-animate(): void
-getReplayStates(): void
sends player input
Server
returns game view
returns game view
<<interface>>
BulletZoneRestClient
setRootUrl(rootUrl: String): void
join(): LongWrapper
grid(): GridWrapper
register(username: String, password:String): BooleanWrapper
login(username: String, password:String): LongWrapper
move(tankId: long, direction: byte): BooleanWrapper
turn(tankId: long, direction: byte): BooleanWrapper
fire(tankId: long): BooleanWrapper
leave(tankId: long): BooleanWrapper
setTankPowerup(tankId:long, powerupValue:Int): void
deploySoldier(tankId: Long): LongWrapper
getHealth(tankId: Long): LongWrapper
updates grid
ReplayGridAdapter
-context: Context
-board: int[][]
+getCount(): int
+getItem(position: int): Object
+getItemId(position: int) : long
+update(newBoard: int[][]): void
+getView(position: int, convertView: View, parent: ViewGroup):View
ClientActivity
-TAG: String
#mGridAdapter: GridAdapter
#gridView: GridView
#healthUpdateTimer: Timer
busProvider: BusProvider
gridPollTask: GridPollerTask
restClient: BulletZoneRestClient
bzRestErrorhandler: BZRestErrorhandler
-tankId: long
-gridEventHandler: Object
previousDirection: byte
tempDirection: byte
#onCreate(savedInstanceState: Bundle): void
#onDestroy(): void
#afterViewInjection(): void
afterInject(): void
joinAsync(): void
+updateGrid(gw: GridWrapper): void
#onButtonMove(view: View): void
moveAsync(tankId: long, direction: byte): void
turnAsync(tankId: long, direction: byte): void
#onButtonFire(): void
leaveGame(): void
performLeave(): void
showConfirmationDialog(): void
login(): void
leaveAsync(): void
-startHeathUpdateTimer():void
-stopHeathUpdateTimer():void
+updateBankBalanceText(numcoins:int): void
#deploySoldier():Void
#deploySoldierAsync():void
+updateTankhealth(health:int):void
+updateHealthAsync(tankId:long):void
gets replay for
updates grid
GridAdapter
-monitor: Object
#inflater: LayoutInflater
-mEntities: int[][]
lastFriendlyDirection: int
lastEnemyDirection: int
+updateList(entities: int[][])
+getCount(): int
+getItem(position: int): Object
+getItem(position: int): Object
+getItemId(position: int): long
+getView(position: int, convertView: View, parent: ViewGroup): View
-isChanged(): Boolean
-writeToFile(): void
+friendlyTank: int
+setFriendlyTank: void
+setEnemyTank: void
+setBullet: void
GridPollerTask
-TAG: String
busProvider: BusProvider
restClient: BulletZoneRestClient
+doPoll(): void
+onGridUpdate(GridWrapper gw): void
polls server
TerrainUI
+friendlyTankImage(imageView: ImageView, direction: int, val: int)
+soldierImage(imageView: ImageView, direction: int, val: int)
+enemyTankImage(imageView: ImageView, direction: int, val: int)
provides grid
gives info

Client Side UML Design Class Diagram

3



Server Side UML Design Class Diagram
GamesController
-log: Logger
-gameRepository: GameRepository
+GamesController(gameRepository: GameRepository)
join(request: HttpServletRequest): ResponseEntity<LongWrapper>
+grid(): ResponseEntity<GridWrapper>
turn(tankId: long, direction: byte): ResponseEntity<BooleanWrapper>
move(tankId: long, direction: byte): ResponseEntity<BooleanWrapper>
fire(tankId: long): ResponseEntity<BooleanWrapper>
fire(tankId: long, bulletType: int): ResponseEntity<BooleanWrapper>
leave(tankId: long): HttpStatus
handleBadRequests(e: Exception): String
sends inputs
sends inputs
receives player input
Client
InMemoryGameRepository
-FIELD_DIM: int
-BULLET_PERIOD: int
-BULLET_DAMAGE: int
-TANK_LIFE: int
-timer: Timer
-idGenerator: AtomicLong
-monitor: Object
-game: Game
-bulletDamage: int[]
-bulletDelay: int[]
-trackActiveBullets: int[]
implements
<<interface>>
GameRepository
join(ip: String): Tank
getGrid(): int[][]
turn(tankId: long, direction: Direction): boolean
move(tankId: long, direction: Direction): boolean
fire(tankId: long, strength: int): boolean
+leave(tankId: long): void
TankLocation
-row: int
-column: int
Stores last location of
GridEvent
-timestamp: Timestamp
-command: String
+getCommand(): String
+getTimestamp(): Timestamp
Has stack of
FieldHolder
-neighbors():Map<Direction, FieldHolder>
-entityHolder: Optional<FieldEntity>
+addNeighbor(direction: Direction, fieldHolder: FieldHolder): void
+isPresent(): boolean
+clearField(): void
Tank
-TAG: String = "Tank";
-id: long
-ip: String
-lastMoveTime: long
-allowedMoveInterval: int
-llastFireTime: long
-allowedFireInterval: int
-numberOfBullets: int
-allowedNumberOfBullets: int
-life: int
-direction: Direction
-isActive: int
ActionCommandInvoker
-history: Stack<GridEvent>
+executeCommand(command: Command): boolean
+getHistory(timestamp: Timestamp): LinkedList<GridEvent>
+getCommandHistory(): Stack<GridEvent>
invokes
<<interface>>
Command
execute(): boolean
getCommandType(): String
implements
commands
ConcreteFireCommand
-action: Action
-bt: int
-oldTankID: long
-newTankID: long
+execute(): boolean
+getCommandType(): String
ConcreteGetHealthCommand
-action: Action
-tankId: long
+execute(): boolean
+getCommandType(): String
ConcreteMoveCommand
-action: Action
-oldDir: Direction
-newDir: Direction
-oldTankID: long
-newTankID: long
+execute(): boolean
+getCommandType(): String
+moveConstraintCheck(): boolean
ConcreteTurnCommand
-action: Action
-oldDir: Direction
-newDir: Direction
-oldTankID: long
-newTankID: long
+execute(): boolean
+getCommandType(): String
+turnConstraintCheck(): boolean
ConcreteUpdateLifeCommand
-action: Action
-tankId: long
-newLife: int
+execute(): boolean
+getCommandType(): String
Game
-FIELD_DIM: int
-id: long
-holderGrid: ArrayList<FieldHolder>
-tanks: ConcurrentMap<Long,Tank>
-playersIP: ConcurrentMap<String,Long>
-soldiers: ConcurrentMap<Long,Soldier>
+getID(): long
+getHolderGrid(): ArrayList<FieldHolder>
+addTank(ip: String, tank: Tank): void
+getTank(tankId: int): Tank
+getTanks(): ConcurrentMap<Long,Tank>
+getGrid(): List<Optional<FieldEntity>>
+getTank(ip: String): Tank
+removeTank(tankId: long): void
+getGrid2D(): int[][]
+addSoldier(ip: String, soldier: Soldier): void
+getTank(tankId: int): Tank
+getSoldiers(): ConcurrentMap<Long,Soldiers>
+getSoldiers(ip: String): Soldier
+removeSoldier(soldierId: long): void
+findTank(tank: Tank, tankId: long): TankLocation
+startEjectionCooldown(): void
+canEject(): boolean
+deploySoldier(tankId: long): LongWrapper
-isValidPosition(x: int, y:int): boolean
populates game board
implements
FieldEntity
{abstract}
#parent: FieldHolder
+getIntValue(): int
+getParent(): FieldHolder
+setParent(parent: FieldHolder): void
+copy(): FieldEntity
+hit(damage: int): void
0..256
Bullet
-tankId: long
-direction: Direction
-damage: int
-bulletId: int
Action
-monitor: Object
-game: Game
-timer: Timer
-bulletDamage: int[]
-bulletDelay: int[]
-trackActiveBullets: int[]
-BULLET_PERIOD: int
+turn(tankId: long, direction: Direction): boolean
+move(tankId: long, direction: Direction): boolean
+fire(tankId: long, strength: int): boolean
inputs to game
Soldier
-TAG: String = "Soldier";
-id: long
-ip: String
-lastMoveTime: long
-allowedMoveInterval: int
-isInTank: boolean
-llastFireTime: long
-allowedFireInterval: int
-numberOfBullets: int
-allowedNumberOfBullets: int
-life: int
-direction: Direction
+reenterTank(tank:Tank):boolean
+startEjectionCooldown(): void
+canEject(): boolean
implements
extends
builds
GameBoardBuilder
gb(): GameBoard
-monitor: Object
-create(): GameBoard
-set(): GameBoard
GameBoard
-FIELD_DIM: int
-monitor: Object
+holderGrid: ArrayList<FieldHolder>
+get(x:int, y:int): FieldHolder
+setEntity(x:int, y:int, f:FieldEntity): void
-createFieldHolderGrid(): void
+add(cell: FieldHolder): void
nukePowerUp
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
applePowerUp
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
Thingamajig
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
Wall
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
Hill
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
Rocky
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string
Forest
Pos:int
destructVal: int
+copy(): FieldEntity
+toString(): string

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.