# Exploding Dogs Final Report

Group 10

Christopher Mach, Gianni Dornevil, Matias Arenas, Rigal Joseph

Dec 3, 2025

## Requirement Specification

1.1 Functional Requirements

Game Setup: The system allows peers to connect to a multiplayer game session without a central server. The system will shuffle the deck and distribute initial cards to all players. The system will put the proper amount of Exploding Kitten and Defuse cards into the deck depending on the number of players.

Gameplay: The system keeps the turn order consistent among all peers. The system allows players to play cards from their hand according to the rules. The system will force players to draw a card at the end of their turn unless skipped or redirected. The system will apply card effects (Attack, Skip, See the Future, Favor, Shuffle, etc.). The system will eliminate a player if they draw an Exploding Kitten and have no Defuse card. The system will declare victory if and when there is one player remaining.

Networking: The system declares player actions (play card, draw card, shuffle, etc.) to all peers. The system synchronizes game state to peers on every action. The system monitors the disconnection of a peer and attempts to keep the game session open.

Error Handling:

Empty Draw Pile

Problem: The player needs to draw a card, but the deck is empty.

Handling: Shuffle the discard pile back into the draw pile.

Incorrect Targeting for Favor/Attack

Problem: The player tries to choose an invalid target (e.g., an eliminated player or themselves).

Handling: Check for valid target credentials and display the message: “Invalid player, choose another.”

Out-of-Turn Actions

Problem: The player tries to play a Nope card at the wrong time.

Handling: Maintain a “current action stack” and only allow valid interrupts. Invalid Nope attempts prevent the player from playing the card.

Data Inconsistency

Problem: The player's hand, draw pile, discard pile, or game state is inconsistent.

Handling: Always update the game state atomically after every turn.

Player Goes Over Turn Time

Problem: The turn timer runs out, and the player hasn't played yet.

Handling: Move on to the next player.

Player Disconnects

Problem: The player loses connection to the game/server.

Handling: If more than one player remains, continue playing and add the disconnected player's cards to the discard pile. If only one player remains, declare them the winner

## 1.2 Essential Use Case.

Use Case: Host Game Session

1. Host selects “Create Game.”
2. System generates a new lobby and awaits connections.
3. Host shares lobby code/IP with friends.
4. Players join, and host starts the game.

Use Case: Join Game

1. Player entesr username
2. System connects client to server and synchronizes game state.
3. Player receives starting hand.

Use Case: Take a Turn

1. Player may play zero or more action cards from hand.
2. Player draws one card from the deck.
3. If the card is an Exploding Dog:
   - If player has Defuse → Defuse card is played, and Exploding Dog is reinserted into the deck.
   - Else → Player is eliminated.
4. Turn passes to next player.

Use Case: Play Action Cards

1. Player selects card from hand (Skip, Attack, Shuffle, See-the-Future, etc.).
2. System validates action.
3. Effect is applied and synchronized across peers.

Variants:

- Attack Card: Skips player’s draw and forces next player to take two turns.
- Skip Card: Ends turn without drawing.

· Shuffle Card: Randomizes deck.

· See-the-Future: Reveals top cards to the player.

Use Case: No More Cards in Deck

1. System detects no more cards in the deck for player to draw from.
2. System takes card from discard pile and reshuffles it to form new draw pile.

Use Case: Win Conditions

1. System detects only one player remains.
2. Game declares winner and display results.

Use Case: Handle Disconnection

1. If a player disconnects, system attempts reconnection.
2. If reconnection fails, remaining peers continue game without them.
3. Disconnected player treated as eliminated.

## Design Specification

2.1 CRC Cards

| **GameCoordinator** | |
|---|---|
| Responsibilities<br>*Create/manage lobby*<br>*Add/remove players*<br>*Start game*<br>*Handle turns*<br>*Validate actions*<br>*Synchronize state across clients*<br>*Handle disconnections*<br>*Determine win conditions* | Collaborators:<br>Game<br>Deck<br>Player<br>ClientHandler<br>Server |

| **Game** | |
|---|---|
| Responsibilities<br>*Manage deck + discard*<br>*Deal cards*<br>*Apply card effects*<br>*Execute turns*<br>*Eliminate players*<br>*Check win state* | Collaborators:<br>GameCooridinator<br>Deck<br>Player<br>Hand<br>Card |

| **Deck** | |
|---|---|
| Responsibilities<br>*Maintain draw pile*<br>*Maintain discard pile*<br>*Shuffle*<br>*Draw cards*<br>*Add effect cards to deck* | Collaborators:<br>Card<br>Game |

| **Player** | |
|---|---|
| Responsibilities<br>*Maintain hand*<br>*Track alive/eliminated state*<br>*Determine valid plays*<br>*Handle drawing* | Collaborators:<br>Deck<br>Card<br>Game<br>Hand<br>GameController |

| **Hand** | |
|---|---|
| Responsibilities<br>*Add/Remove cards*<br>*Validate hand actions* | Collaborators:<br>Player<br>Card |

| **Card** | |
|---|---|
| Responsibilities<br>*Represent card type* | Collaborators:<br>Game |

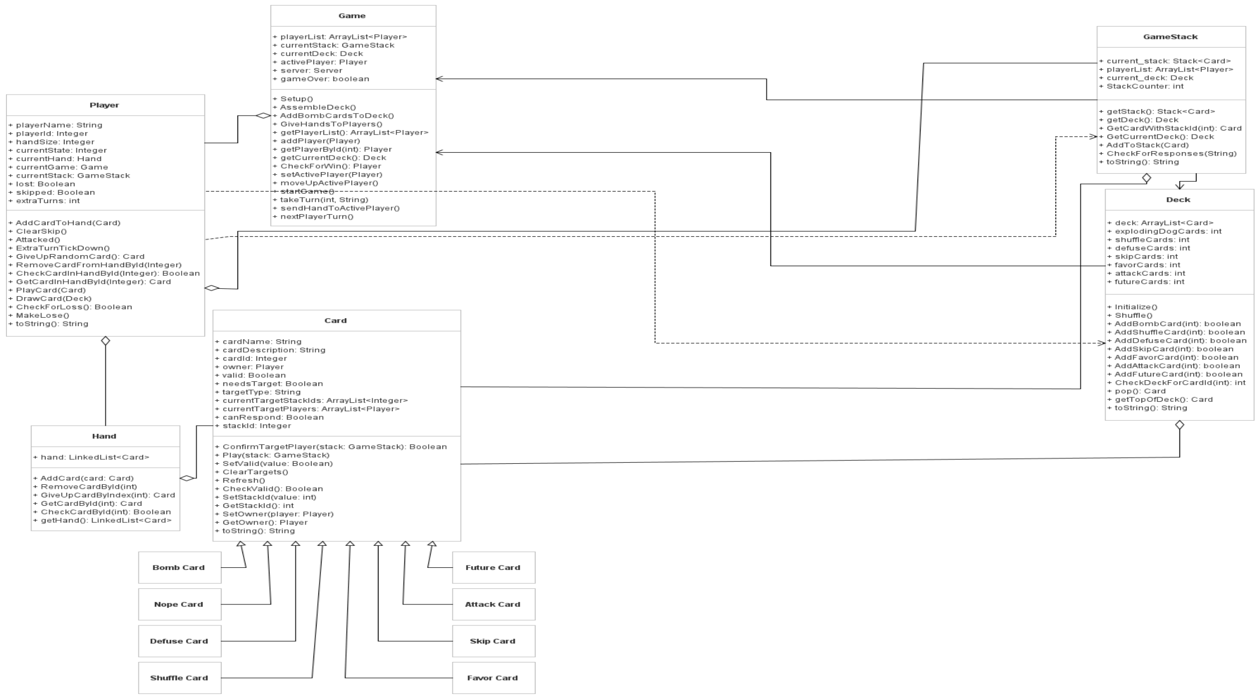
| | |
|---|---|
| *Define card rules* | Deck<br>Player |

| ClientHandler | |
|---|---|
| <u>Responsibilities</u><br>*Maintain connection with one client*<br>*Receive commands*<br>*Send updates  Reconnect attempts* | <u>Collaborators:</u><br>Server<br>GameCoordinator<br>Player |

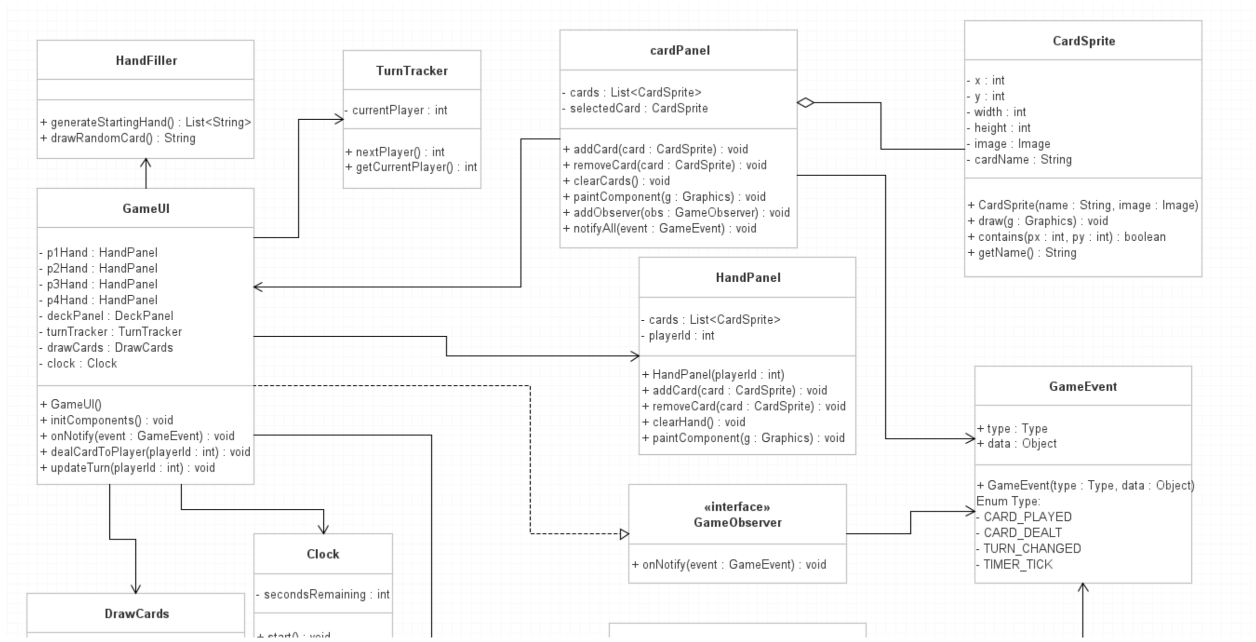
| Server | |
|---|---|
| <u>Responsibilities</u><br>*Start listening socket*<br>*Accept clients*<br>*Spawn ClientHandler* | <u>Collaborators:</u><br>ClientHandler<br>GameCoordinator |

2.2 UML Class Diagrams

Game Class Diagram:

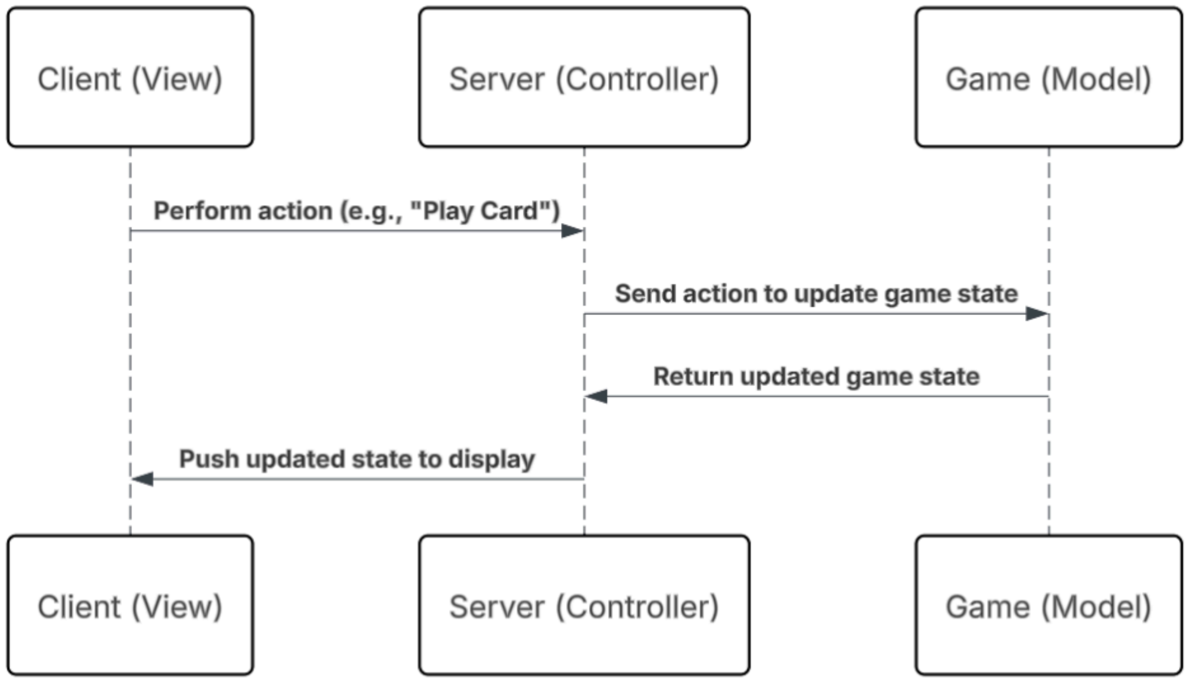


Game UI class diagram:

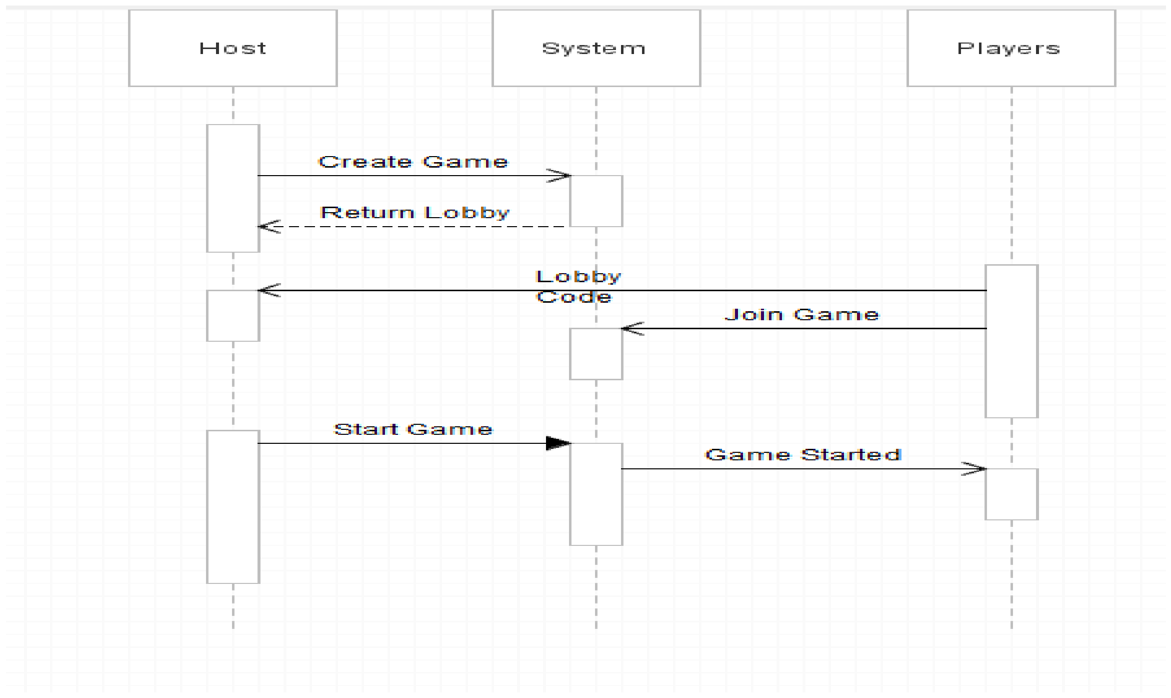

HandFiller
+ generateStartingHand() : List<String>
+ drawRandomCard() : String
TurnTracker
- currentPlayer : int
+ nextPlayer() : int
+ getCurrentPlayer() : int
cardPanel
- cards : List<CardSprite>
- selectedCard : CardSprite
+ addCard(card : CardSprite) : void
+ removeCard(card : CardSprite) : void
+ clearCards() : void
+ paintComponent(g : Graphics) : void
+ addObserver(obs : GameObserver) : void
+ notifyAll(event : GameEvent) : void
CardSprite
- x : int
- y : int
- width : int
- height : int
- image : Image
- cardName : String
+ CardSprite(name : String, image : Image)
+ draw(g : Graphics) : void
+ contains(px : int, py : int) : boolean
+ getName() : String
GameUI
- p1Hand : HandPanel
- p2Hand : HandPanel
- p3Hand : HandPanel
- p4Hand : HandPanel
- deckPanel : DeckPanel
- turnTracker : TurnTracker
- drawCards : DrawCards
- clock : Clock
+ GameUI()
+ initComponents() : void
+ onNotify(event : GameEvent) : void
+ dealCardToPlayer(playerId : int) : void
+ updateTurn(playerId : int) : void
HandPanel
- cards : List<CardSprite>
- playerId : int
+ HandPanel(playerId : int)
+ addCard(card : CardSprite) : void
+ removeCard(card : CardSprite) : void
+ clearHand() : void
+ paintComponent(g : Graphics) : void
GameEvent
+ type : Type
+ data : Object
+ GameEvent(type : Type, data : Object)
Enum Type:
- CARD_PLAYED
- CARD_DEALT
- TURN_CHANGED
- TIMER_TICK
«interface»
GameObserver
+ onNotify(event : GameEvent) : void
Clock
- secondsRemaining : int
DrawCards

2.3 Sequence Diagrams. One diagram for each scenario.

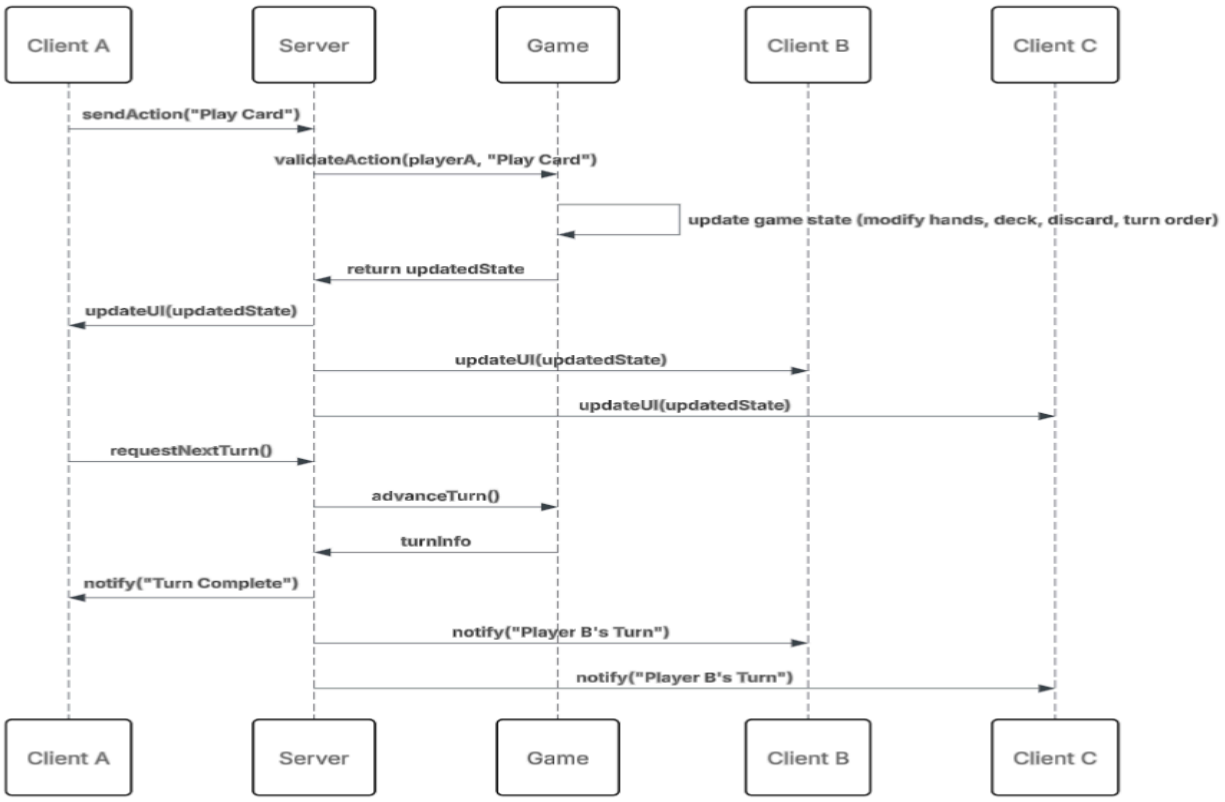
MVC Sequence Diagram:


Client (View)
Server (Controller)
Game (Model)
Perform action (e.g., "Play Card")
Send action to update game state
Return updated game state
Push updated state to display
Client (View)
Server (Controller)
Game (Model)

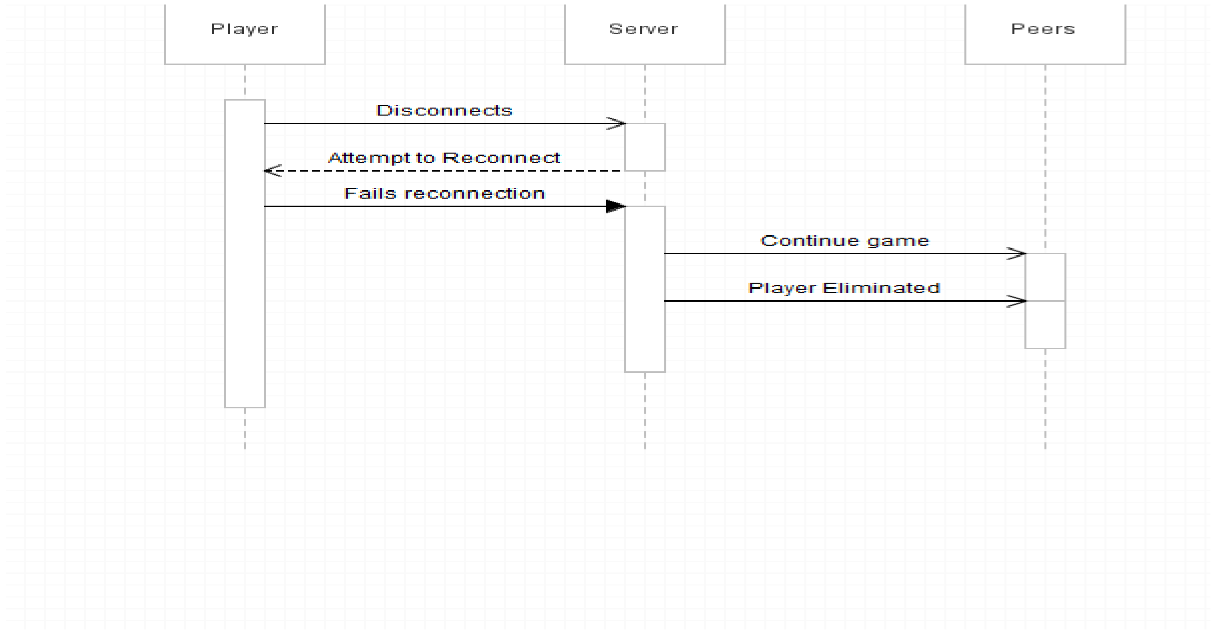
Hosting Game:



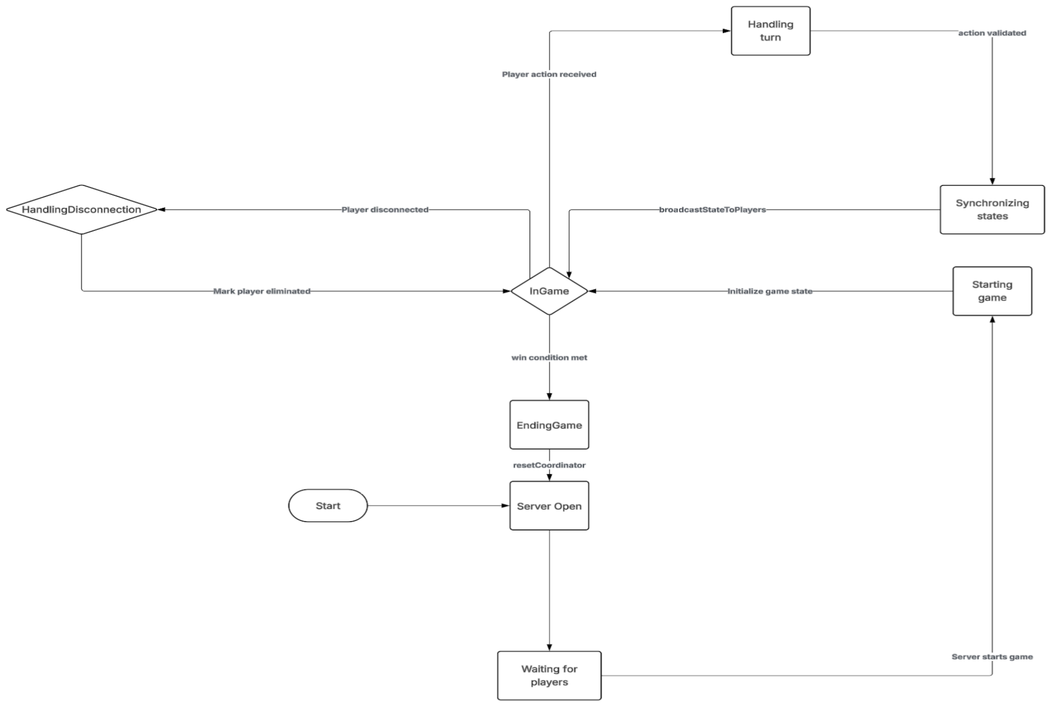
Game Interaction:



Server Disconnection:



2.4 State Diagram(s). Necessary for classes with non-trivial behavior, such as a GUI controller that controls the sequence in which windows are displayed.


Handling turn
action validated
Player action received
HandlingDisconnection
Player disconnected
broadcastStateToPlayers
Synchronizing states
InGame
Mark player eliminated
Initialize game state
Starting game
win condition met
EndingGame
resetCoordinator
Start
Server Open
Waiting for players
Server starts game