Eg

The LUNA GAME

1. Problem Statement

Traditional dice games like *Pop & Krepp* suffer from disputes due to unclear dice rolls (e.g., dice stacking, unstable positions) and lack of structured rule enforcement. These disputes disrupt the flow, cause conflicts between players, and make the game less fair and enjoyable.

The goal is to develop **LUNA**, a **digital multiplayer dice game** with clear, automated rules and a **host (Knoksman)** to manage the game, ensuring fairness, real-time play, and an online betting or point system.

---

2. Objectives

- Eliminate physical conflicts through **automated dice rolls**.
- Provide a **multiplayer online platform** for fair play.
- Allow a **Knoksman** to host games, manage bets, and track winnings/losses.
- Maintain **real-time updates** so all players see dice rolls instantly.
- Support both **betting** and **point-based scoring modes**.

---

3. Functional Requirements

- **User Management:** Players and Knoksman accounts with login/logout.
- **Game Hosting:** Knoksman creates sessions; players join via a code or link.
- **Dice Rolling System:** Automated dice roll generator for fairness.
- **Game Rules Engine:** Handles Pop, Krepp, Mail, and Uppen logic automatically.
- **Score/Bet Tracking:** Keeps track of player bets, wins, and losses.
- **Real-Time Play:** WebSockets for instant dice roll updates to all players.

---

4. Non-Functional Requirements

- **Performance:** Real-time dice results for all connected players.
- **Security:** Safe handling of user accounts, bets, and results.
- **Scalability:** Ability to handle multiple games at once.
- **Usability:** Simple UI for both mobile and web players.

---

5. Proposed Architecture

Eg

Frontend (Client Side):

- React.js (Web) / React Native (Mobile)
- Displays game board, dice rolls, scores, and bets.

Backend (Server Side):

- Node.js or Python Flask/Django
- Handles game logic, player management, and real-time communication.

Database:

- Firebase (real-time) or MySQL/PostgreSQL
- Stores game data, user profiles, and transaction history.

Real-Time Communication:

- WebSockets or Firebase Realtime Database for live dice updates.

---

6. Technology Stack

Component	**Technology**
Frontend	React.js / React Native
Backend	Node.js (Express.js + Socket.IO) / Flask / Django
Database	Firebase / MySQSL
Real-Time Updates	WebSockets / Firebase
Hosting	AWS / Render / Heroku

---

7. Development Phases

1. **Requirement Analysis** → Gather all rules, player behaviors, and features.
2. **System Design** → Create architecture diagrams, data flow diagrams, and UI mockups.
3. **Implementation** → Develop frontend, backend, and database components.
4. **Integration & Testing** → Ensure all parts work together smoothly.
5. **Deployment** → Host the application on a cloud platform.
6. **Maintenance** → Fix bugs, add features, and improve user experience

Eg

1. Use Case Diagram (UML)

Purpose:

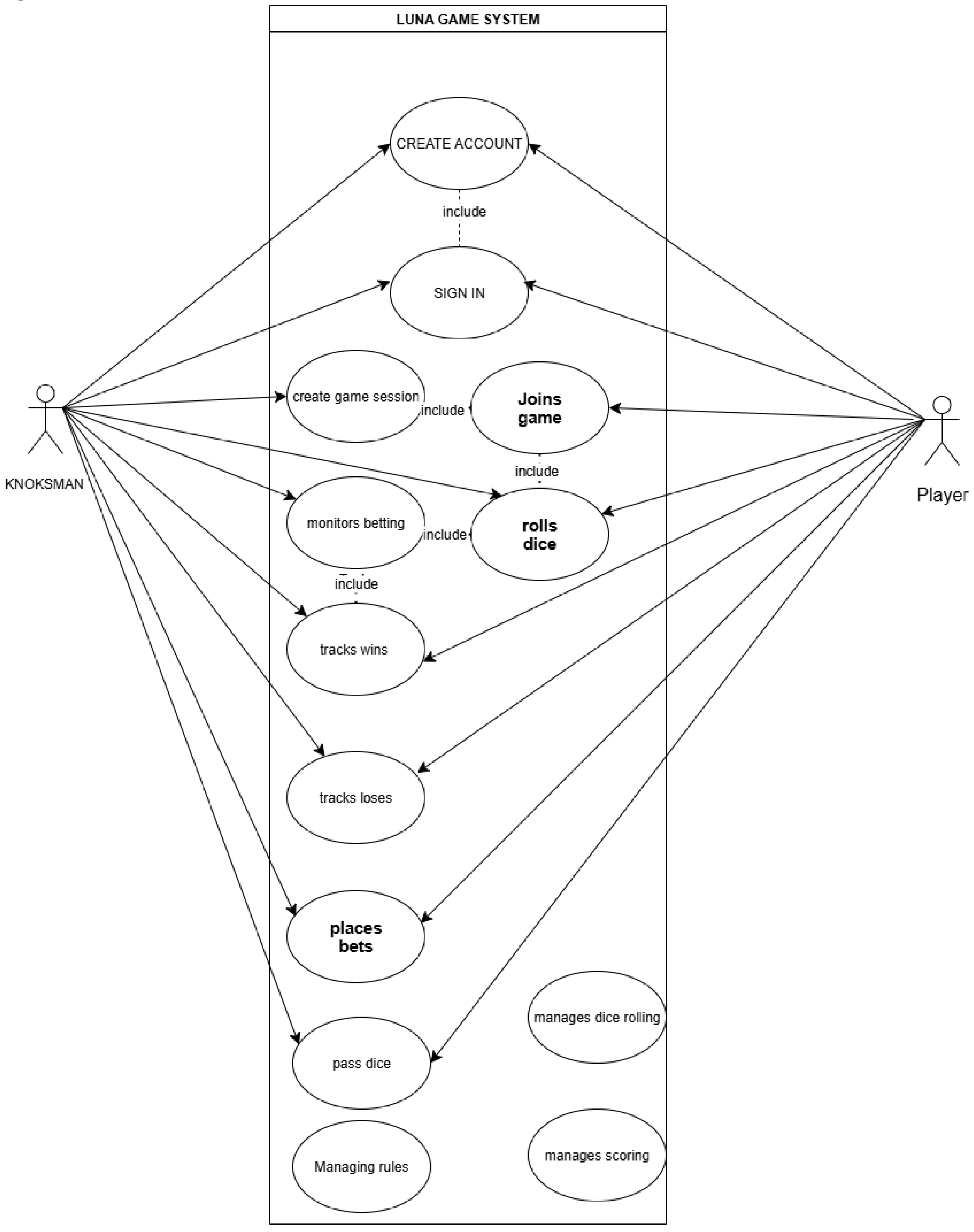
Shows the actors (e.g., Player, Knoksman, System) and the interactions with the system.

Actors:

- **Player → Joins game, rolls dice, places bets, wins/losses.**
- **Knoksman → Creates game session, monitors betting, tracks wins/losses.**
- **System → Manages dice rolling, rules, and scoring.**

Eg

Eg


LUNA GAME SYSTEM
CREATE ACCOUNT
include
SIGN IN
create game session
include
Joins game
include
rolls dice
monitors betting
include
include
tracks wins
tracks loses
places bets
manages dice rolling
pass dice
Managing rules
manages scoring
KNOKSMAN
Player

Eg

2. Activity Diagram

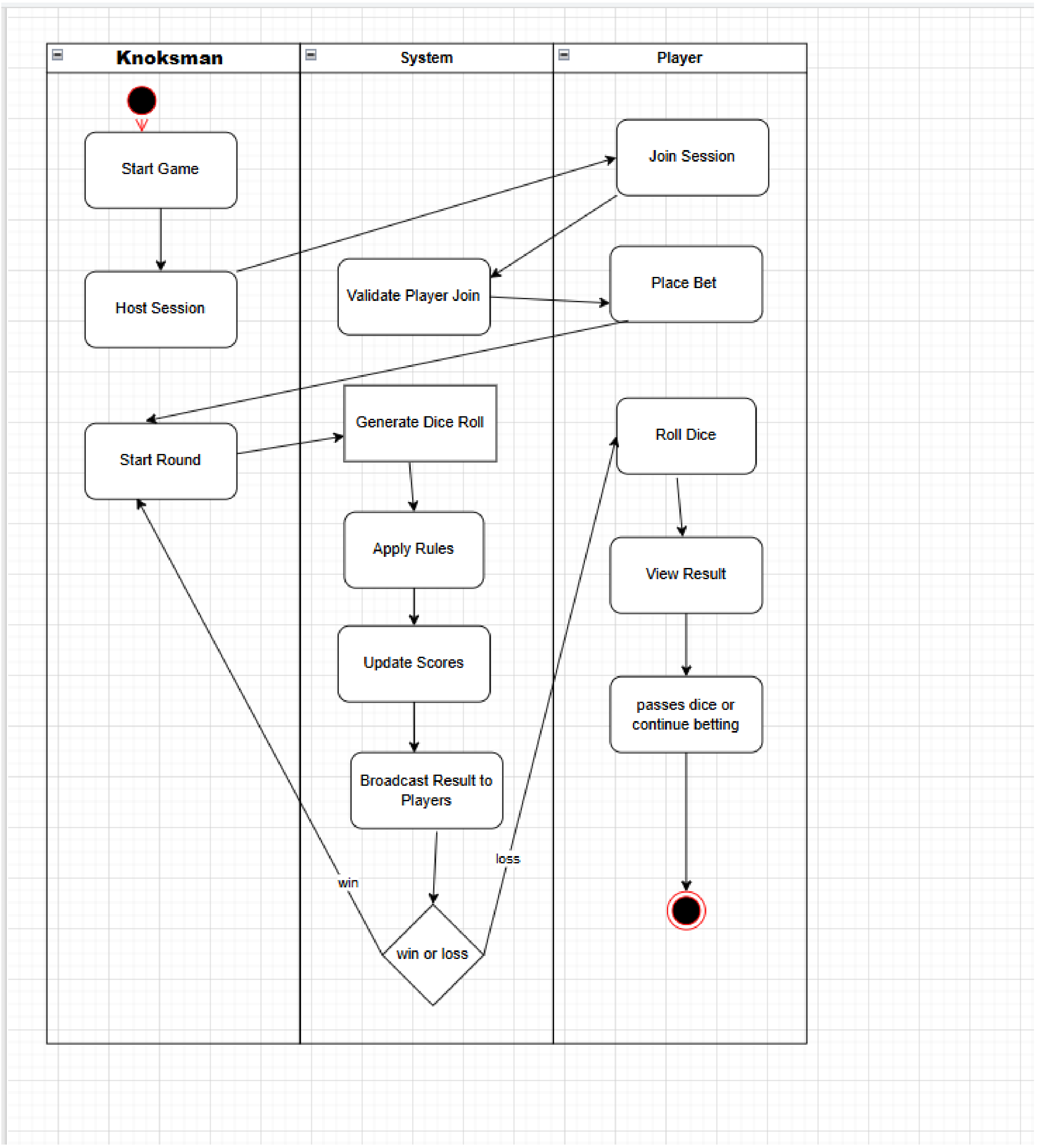
Purpose:

Represents the workflow of the game process.

Flow:

1. **Start Game → Player joins → Place Bet → Roll Dice**
2. **Check result:**
   - **Pop (7/11) → Win → Update Scores**
   - **Krepp (2/3/12) → Loss → Opponent Wins**
   - **Mail → Set Point → Keep rolling until Mail → Win or 7 → Loss**

Eg



E

3. Business Process Model (BPM)

Purpose:

Shows the business logic:

- **Knoksman initiates game → Players place bets → Dice rolls handled by system → Results declared → Scores updated → Next round starts.**

Eg

4. State Machine Diagram

Purpose:

Shows state changes of the game as it progresses.

States:

- **Waiting for Players**
- **Bet Placed**
- **Dice Rolling**
- **Pop Win**
- **Krepp Loss**
- **Mail Set → Rolling until Point or 7**
- **Game Over / Next Round**

---

5. Sequence Diagram

Purpose:

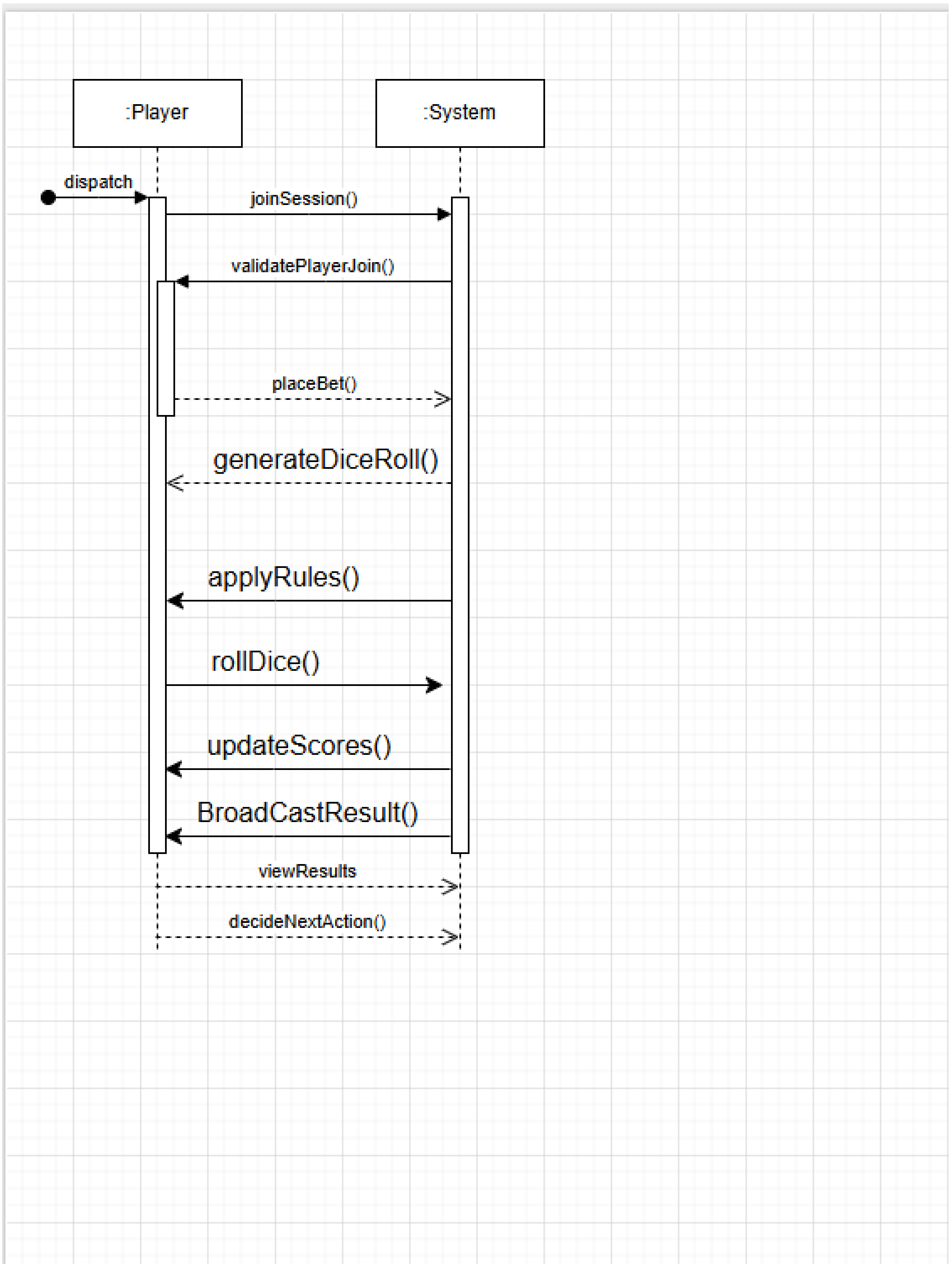
Represents the order of interactions between system components.

Actors: Player, Knoksman, Game System
Sequence:

1. **Player joins game**
2. **Knoksman starts game**
3. **Player places bet → Dice Roll request → System processes → Result sent → Scores updated**

Eg

Eg



6. Entity Class Diagram

Purpose:

Shows database entities and relationships.

Entities:

Eg

- **Player (PlayerID, Name, Score, Balance)**
- **GameSession (SessionID, HostID, Status)**
- **Bet (BetID, PlayerID, Amount, Result)**
- **DiceRoll (RollID, SessionID, Value1, Value2, ResultType)**
- **Knoksman (HostID, Name, Earnings)**

Relationships:

- **Player → Places → Bet**
- **Bet → Linked to → GameSession**
- **GameSession → Has → DiceRolls**
- **Knoksman → Hosts → GameSession**

Pop & Krepp Game Rules (Using a pair of dice)

1. Key Terms

- **Pop** → Rolling **7** or **11** with two dice on the first roll = **Instant Win**
- **Krepp** → Rolling **2**, **3**, or **12** on the first roll = **Instant Loss**
- **Mail** → Any other number (4, 5, 6, 8, 9, 10) becomes the **point number**
- **Uppen** → The **opposite number** of the Mail on the dice.
  Example: If Mail = 4, Uppen = 10 because it is the opposite side of the dice numbers.

---

2. Game Flow

1. **Start**:
   - A player places a bet (this is called "setting").
   - The **Knoksman** (like the dealer) oversees bets and starts the game.
2. **First Roll**:
   - **Pop (7 or 11)** → Player **wins immediately**; opponent resets.
   - **Krepp (2, 3, 12)** → Player **loses immediately**; must set again or pass dice.
   - **Mail (4, 5, 6, 8, 9, 10)** → The number rolled becomes the **point**.
3. **Second Phase (if Mail was rolled)**:
   - Player keeps rolling until:
     - They hit the **Mail** again → **Win**

Eg

        - They roll a **7** → **Loss** (opponent wins)

4. **Uppen Rule**:
    - If the player declares **Uppen** (opposite number of Mail) and rolls it → **Win**
    - If they roll **Pop (7 or 11)** in this phase → Still a **Win**

---

3. Knoksman Role

- Keeps track of winnings.
- If a player wins **three times in a row**, Knoksman takes **amount specified for the game but is not complasary for the player to keep setting it**
- Knoksman can also start the game. If he loses, he must set again like other players.

---

4. Summary Table

Roll Outcome	**Result**
7 or 11 (first roll)	**Pop → Win**
2, 3, 12 (first roll)	**Krepp → Loss**
Mail (4,5,6,8,9,10)	Set Point → Roll again until Mail = Win or 7 = Loss
Uppen of Mail	**Win**
7 after Mail	**Loss**