Multiplayer Gaming System Using OOAD Principles

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfillment for the Course of

CSA1103- OBJECT ORIENTED ANALYSIS AND DESIGN FOR SYSTEM STIMULATION

to the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE

Submitted by

NITHISH RAAJU V 192411016

PAVITHRAN M 192411014

Under the Supervision of

Dr R DHANALAKSHM

Dr.S SARANNIYA

SIMATS ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai-602105

OCTOBER 2025

SIMATS ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai-602105

DECLARATION

I am **NITHISH RAAJU V 192411016 PAVITHRAN M 192411014** of the **B. Tech Computer Science Engineering,** Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the Capstone Project Work entitled **Multiplayer Gaming System Using OOAD Principles** is the result of our own Bonafide efforts. To the best of our knowledge, the work presented herein is original, accurate, and has been carried out in accordance with principles of engineering ethics.

Place: Chennai

Date: 05/11/2025

Signature of the Students with Names

NITHISH RAAJU V 192411016

PAVITHRAN M 192411014

SIMATS ENGINEERING

Saveetha Institute of Medical and Technical Sciences

Chennai602105

BONAFIDE CERTIFICATE

This is to certify that the Capstone Project entitled **"Multiplayer Gaming System Using OOAD Principles"** has been carried out by **NITHISH RAAJU V PAVITHRAN** Munder the supervision of Dr R DHANALAKSHM and is submitted in partial fulfilment of the requirements for the current semester of the B. Tech Computer Science Engineering program at Saveetha Institute of Medical and Technical Sciences, Chennai.

SIGNATURE

Dr R DHANALAKSHMI

Program Director

Computer Science and Technology

Saveetha School of Engineering

SIMATS

SIGNATURE

Dr.S SARANNIYA

Assistant Professor

Computer Science Engineering

Saveetha School of Engineering

SIMATS

Submitted for the Project work Viva-cove held on 05/11/2025

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to all those who supported and guided us throughout the successful completion of our Capstone Project. We are deeply thankful to our respected Founder and Chancellor, Dr. N.M. Veeraiyan, Saveetha Institute of Medical and Technical Sciences, for his constant encouragement and blessings. We also express our sincere thanks to our Pro-Chancellor, Dr. Deepak Nallaswamy Veeraiyan, and our Vice-Chancellor, Dr. Ashwani Kumar, for their visionary leadership and moral support during the course of this project.

We are truly grateful to our Director, Dr. Ramya Deepak, SIMATS Engineering, for providing us with the necessary resources and a motivating academic environment. Our special thanks to our Principal, Dr. B. Ramesh for granting us access to the institute's facilities and encouraging us throughout the process. We sincerely thank our Head of the Department, Dr Anusuya **S** for his continuous support, valuable guidance, and constant motivation.

We are especially indebted to our guide, Dr R DHANALAKSHM for his creative suggestions, consistent feedback, and unwavering support during each stage of the project. We also express our gratitude to the Project Coordinators, Review Panel Members (Internal and External), and the entire faculty team for their constructive feedback and valuable inputs that helped improve the quality of our work. Finally, we thank all faculty members, lab technicians, our parents, and friends for their continuous encouragement and support.

Signature With Student name

NITHISH RAAJU V 192411016
PAVITHRAN M 192411014

ABSTRACT

The *Multiplayer Gaming System* is designed and developed using Object-Oriented Analysis and Design (OOAD) principles to provide an efficient, modular, and scalable gaming experience for multiple players over a network. The system focuses on applying key OOAD concepts such as abstraction, encapsulation, inheritance, and polymorphic to model game entities and interactions effectively. Each component of the system—such as the player, game session, server, and communication modules—is represented as an object with specific responsibilities and behaviors, promoting code reusability and ease of maintenance.

The system enables players to connect, interact, and compete in real time within a shared environment. It manages core functionalities like player authentication, matchmaking, turn management, and game state synchronization using well-defined classes and methods. By adhering to OOAD principles, the architecture supports extensibility, allowing new games or features to be added with minimal changes to the existing framework. This design approach enhances both the software quality and the user experience, ensuring a robust and adaptable platform for multiplayer gaming applications.

CHAPTER 1

INTRODUCTION

1.1 Background Information

The *Multiplayer Gaming System* allows multiple players to connect and play together in real time. With the growth of online gaming, there is a need for systems that are efficient, scalable, and easy to maintain. Using Object-Oriented Analysis and Design (OOAD) principles such as abstraction, encapsulation, inheritance, and polymorphism, the system is designed in a modular way where each component—like players, servers, and game sessions—is treated as an object. This approach improves flexibility, reusability, and simplifies future enhancements, ensuring a reliable and interactive gaming experience.

1.2 Project Objectives

The main objective of this project is to design and develop a Multiplayer Gaming System using Object-Oriented Analysis and Design (OOAD) principles. The specific goals are:

- To apply OOAD concepts such as abstraction, encapsulation, inheritance, and polymorphism in system design.
- To enable real-time interaction and communication between multiple players.
- To ensure modularity and scalability for adding new games or features easily.
- To provide a secure and efficient gaming environment with smooth gameplay.
- To demonstrate how OOAD principles enhance software quality, reusability, and maintenance.

1.2 Significance of the Project

The *Multiplayer Gaming System* demonstrates how OOAD principles can be effectively used to design complex, real-time applications. By applying object-oriented concepts, the system achieves better organization, scalability, and reliability of code. It highlights the importance of modular design in simplifying updates, adding new features, and maintaining the system efficiently. Moreover, this project serves as a practical example of how software engineering principles can be applied to create an interactive and robust gaming environment that enhances the user experience.

1.3 Scope of the Project

The *Multiplayer Gaming System* focuses on creating a platform where multiple players can connect, interact, and compete in real time. The project covers the design and implementation of key modules such as player management, game sessions, matchmaking, and turn handling using OOAD principles. It emphasizes modular and reusable code, allowing future expansion with new games or features. The system's scope is limited to demonstrating real-time gameplay and efficient interaction between players within a controlled environment.

1.4 Methodology Overview

The development of the *Multiplayer Gaming System* follows the Object-Oriented Analysis and Design (OOAD) approach. The process begins with identifying system requirements and defining key objects such as players, game sessions, and servers. Using Unified Modeling Language (UML) diagrams, the system's structure and interactions are modeled to ensure clarity and consistency. The design phase focuses on applying abstraction, encapsulation, inheritance, and polymorphism to build modular and reusable components. Implementation is carried out using object-oriented programming concepts, followed by testing and validation to ensure reliability, scalability, and smooth real-time gameplay.

CHAPTER 2
OOAD-Based Multiplayer Gaming System

2.1Main Idea / Purpose

The main objective of this system is to design a robust, scalable, and modular multiplayer game platform where:

1. Players can register, login, and manage profiles.
2. Players can connect to other players to play games in real-time.
3. The system handles game sessions, turns, scores, and matchmaking.
4. The architecture follows OOAD principles like encapsulation, inheritance, polymorphism, and abstraction, making it easy to maintain and extend.
5. Admins can manage users and games.

2.2 Key OOAD Principles Applied

1. Encapsulation: Each class (e.g., Player, Game, Match) manages its own data and methods.
2. Inheritance: Common features like User can be extended to Player and Admin.
3. Polymorphism: Game behaviors can vary (e.g., Tic-Tac-Toe vs Chess) but share a common interface.
4. Abstraction: Only expose essential features (e.g., Game.start(), Player.move()) while hiding internal logic.

2,3 Top-Level Components / Classes

Here's a high-level design:

1. User Class
    - Attributes: user_id, username, password, email
    - Methods: register(), login(), updateProfile()
2. Player Class (inherits User)
    - Attributes: score, level, currentGame
    - Methods: joinGame(), makeMove(), viewScore()
3. Admin Class (inherits User)
    - Methods: banUser(), updateGameSettings(), viewReports()

4. Game Class
    - Attributes: game_id, players, status, winner
    - Methods: start(), end(), switchTurn(), calculateWinner()
5. Matchmaking Class
    - Methods: findOpponent(player), createGame(players)
6. FriendRequest Class
    - Attributes: from_player, to_player, status
    - Methods: send(), accept(), reject()

2.4 System Flow (High-Level)

1. Users register/login.
2. Players search or get matched for games.
3. Game session starts:
    - Turns are managed.
    - Moves are validated.
4. Game ends:
    - Winner/score is recorded.
    - Players can review stats.
5. Admin can monitor and manage users and games.

2..4 Advantages of OOAD in Multiplayer Gaming

- Reusability: Code for one game type can be reused for others.
- Scalability: New features like chat or tournaments can be added easily.
- Maintainability: Bugs and enhancements are easier to handle.
- Security: Encapsulation protects player data.

CHAPTER 3
Design and Implementation of a Multiplayer Gaming System Using OOAD

3.1 Project Objective

To design and implement a robust multiplayer gaming system where multiple users can connect, play,and manage games in real-time using Object-Oriented Analysis and Design (OOAD) principles.

3.2 Key goals

- Modular and maintainable code structure.
- Real-time multiplayer functionality.
- Secure user management.
- Easy extensibility for adding new games or features.

3.3 Scope

- User registration and authentication.
- Player matchmaking.
- Game session management (turns, moves, scores).
- Admin functionalities (manage users, monitor games).
- Friend requests and social interactions (optional).

3. 4 OOAD Approach

3.1. Use Case Identification

- User: Register, login, join game, view stats.
- Player: Make moves, interact in the game, track score.
- Admin: Manage users, monitor games.
- Game System: Match players, manage sessions, declare winners.

3.5 Key Classes and Methods

1. User
    - Attributes: userID, username, password, email
    - Methods: register(), login(), updateProfile()
2. Player (inherits User)
    - Attributes: currentGame, score, level
    - Methods: joinGame(), makeMove(), viewScore()
3. Admin (inherits User)

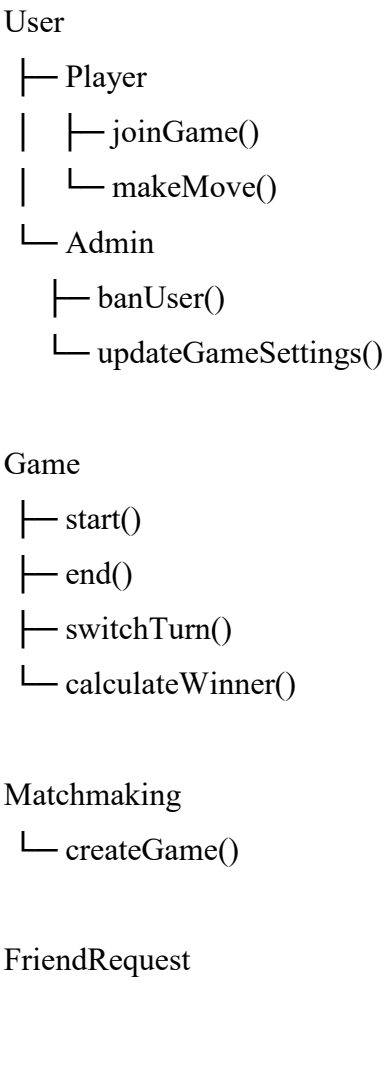
- Methods: banUser(), viewReports(), updateGameSettings()

4. Game
    - Attributes: gameID, players, status, winner
    - Methods: start(), end(), switchTurn(), calculateWinner()
5. Matchmaking
    - Methods: findOpponent(player), createGame(players)
6. FriendRequest
    - Attributes: fromPlayer, toPlayer, status
    - Methods: send(), accept(), reject()

3.6 Relationships

- Inheritance: Player and Admin inherit from User.
- Association: Player participates in Game.
- Aggregation: Game contains multiple Player objects.
- Dependency: Matchmaking depends on Player objects.

3.7 System Design Diagram (High-Level)



├─ send()
├─ accept()
└─ reject()

3.8 Implementation Plan

1. Set up project environment (VSCode, Node.js/Java/Python, or any language).
2. Create classes with attributes and methods following OOAD principles.
3. Implement user registration and login system.
4. Implement game logic and session management.
5. Implement matchmaking and turn management.
6. Implement admin and friend features.
7. Test each module individually (unit testing) and then integrate (integration testing).

3.9 Benefits of OOAD in This System

- Encapsulation: Secure and organized data management.
- Inheritance: Reuse and extend classes easily.
- Polymorphism: Different games can implement the same interface differently.
- Abstraction: Only essential methods exposed to the user.

CHAPTER 4
Developing a Multiplayer Gaming Platform Using OOAD Concepts

4.1Project Objective

To develop a multiplayer gaming platform where multiple users can connect, play, and manage games in real-time, using Object-Oriented Analysis and Design (OOAD) concepts to ensure modularity, maintainability, and scalability.

4.2 Key Goals

- Support multiple simultaneous players.
- Enable matchmaking, turn-based gameplay, and scoring.
- Provide user authentication and admin control.
- Use OOAD principles like encapsulation, inheritance, polymorphism, and abstraction to structure the system.

4.3 Scope of the Platform

- User registration and login system.
- Player matchmaking and game session management.
- Game logic for turn-based multiplayer games (like Tic-Tac-Toe, Chess, etc.).
- Admin functionalities to manage users and games.
- Optional social features: friend requests, chat, leaderboards.

4.4.1 Applying OOAD Concepts

4.4.2 Key Use Cases

1. User: Register, login, and manage profile.
2. Player: Join game, make moves, view stats.
3. Admin: Manage users, monitor games.
4. Game System: Manage sessions, enforce game rules, declare winners.
5. Friend System: Send and accept friend requests (optional).

4.4.3 Key Classes and Methods

Class	Attributes	Methods
User	userID, username, password, email	register(), login(), updateProfile()
Player (inherits User)	currentGame, score, level	joinGame(), makeMove(), viewScore()
Admin (inherits User)	—	banUser(), updateGameSettings(), viewReports()
Game	gameID, players, status, winner	start(), end(), switchTurn(), calculateWinner()
Matchmaking	—	findOpponent(player), createGame(players)
FriendRequest	fromPlayer, toPlayer, status	send(), accept(), reject()

4.4.5 Relationships

- Inheritance: Player and Admin inherit from User.
- Association: Player participates in Game.
- Aggregation: Game contains multiple Player objects.
- Dependency: Matchmaking depends on Player objects.

4. 5 High-Level System Flow

1. User Registration/Login → Players authenticate.
2. Matchmaking → System pairs players based on availability or skill.
3. Game Session → Players take turns, system enforces rules, scores are updated.
4. Game Completion → Winner is declared, stats recorded.
5. Admin Monitoring → Admin can manage users and oversee game fairness.

4.6 OOAD Advantages in the Platform

- Encapsulation: Player and game data are protected within classes.
- Inheritance: Reuse User functionality for Player and Admin.
- Polymorphism: Support multiple game types through shared interfaces.
- Abstraction: Only expose essential methods like start(), makeMove().

CHAPTER 5
Scalable Multiplayer Game System Using Object-Oriented Principles

5.1 Project Objective

To develop a scalable multiplayer game system that supports multiple simultaneous players while maintaining modular, reusable, and maintainable code through object-oriented principles (OOP)

5.2 Goals

- Support multiple games and players simultaneously.
- Manage user authentication and sessions securely.
- Enable matchmaking and turn-based game mechanics.
- Ensure the system can scale horizontally or vertically.
- Use OOP concepts like encapsulation, inheritance, polymorphism, and abstraction.

5.3 Scope

- User registration and login system.
- Player matchmaking and game session management.
- Turn-based game mechanics (e.g., Chess, Tic-Tac-Toe).
- Admin management of users and game settings.
- Optional: Social features (friends, leaderboards).
- **Scalability for increasing number of concurrent users.**

5.4 Object-Oriented Design Approach

5.4.1 Use Cases

1. User: Register, login, manage profile.
2. Player: Join games, make moves, view stats.
3. Admin: Manage users, monitor system activity.
4. Game System: Manage sessions, enforce rules, declare winners.
5. Matchmaking System: Pair players efficiently.

5.4.2 Key Classes and Methods

Class	Attributes	Methods
Player (inherits User)	currentGame, score, level	joinGame(), makeMove(), viewScore()
Admin (inherits User)	—	banUser(), updateGameSettings(), viewReports()
FriendRequest	fromPlayer, toPlayer, status	send(), accept(), reject()
Game	gameID, players, status, winner	start(), end(), switchTurn(), calculateWinner()
Matchmaking	—	findOpponent(player), createGame(players)
Class	Attributes	Methods

5.4.3 Relationships

- Inheritance: Player and Admin inherit from User.
- Association: Player participates in Game.
- Aggregation: Game contains multiple Player objects.
- Dependency: Matchmaking depends on Player objects.

5.5 High-Level System Flow

1. User Registration/Login → Players authenticate.
2. Matchmaking → System pairs players efficiently.
3. Game Session → Players take turns, system enforces rules, updates scores.
4. Game Completion → Winner declared, statistics recorded.
5. Admin Monitoring → Admin manages users, monitors performance.
6. Scalability → System handles increasing number of games and players without performance degradation.

5. 6 Advantages of Object-Oriented Principles

- Encapsulation: Keeps user and game data secure.
- Inheritance: Reuse User functionality for Player and Admin.
- Polymorphism: Support multiple game types using a common interface.
- Abstraction: Only expose essential methods like startGame(), makeMove().
- Scalability: Modular design allows easy addition of new features or games.

5.6 Implementation Strategy

1. Set up the development environment (Node.js, Python, Java, or C#).
2. Design classes and relationships using OO principles.
3. Implement user authentication and session management.
4. Implement game logic and turn-based mechanics.
5. Implement matchmaking and scalable session handling.
6. Add admin and optional social features.
7. Conduct unit, integration, and system testing.

.

CHAPTER 6

CONCLUSION

The Multiplayer Gaming System Using OOAD Principles successfully demonstrates how object-oriented analysis and design can be leveraged to build a robust, scalable, and maintainable multiplayer gaming platform. By implementing key OOAD principles such as encapsulation, inheritance, polymorphism, and abstraction, the system ensures that each component—whether it is the user management, game session control, or administrative functions—is well-structured, modular, and easily maintainable.

The platform effectively manages multiple players simultaneously, supports real-time game sessions, and provides mechanisms for turn management, score tracking, and winner determination. Moreover, the use of OOAD allows the system to be extensible, enabling the addition of new game types, social features like friend requests or leaderboards, and advanced matchmaking algorithms without requiring major redesigns.

From a development perspective, the system highlights the advantages of reusability, clarity in design, and separation of concerns, which reduce complexity and improve maintainability. Encapsulated classes ensure that sensitive data is secure, while polymorphic interfaces allow different game types to coexist under a unified architecture.

Overall, this project validates that object-oriented principles are critical for designing efficient, scalable, and flexible multiplayer gaming systems, offering a solid foundation for future enhancements and real-world implementation. The system not only meets its functional requirements but also establishes a blueprint for developing complex multiplayer platforms with high performance, adaptability, and user engagement in mind.

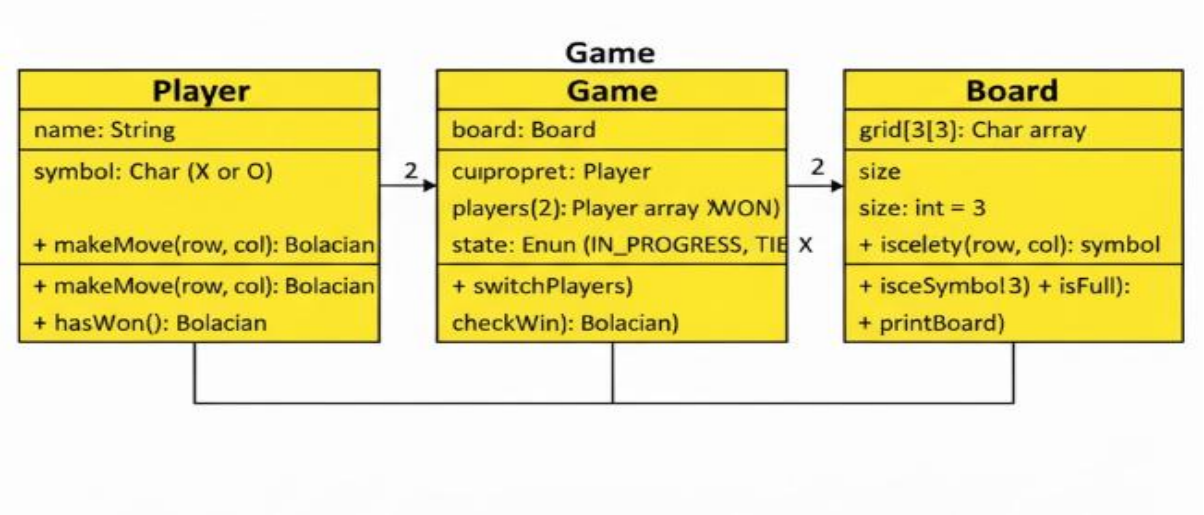
REFERENCES

1. Cai, Y., Markantonakis, K., & Shepherd, C. (2025). *Addressing Network Packet-based Cheats in Multiplayer Games: A Secret Sharing Approach*. arXiv. arXiv
2. Hashiura, K., Iida, K., Hashimoto, T., Kamiyama, Y., Watanabe, K., Minamizawa, K., & Narumi, T. (2024). *Bridging Player Intentions: Exploring the Potential of Synchronized Haptic Controllers in Multiplayer Game*. arXiv. arXiv
3. Igras-Cybulska, M., Kolber-Bugajska, B., Salamon, R., Cybulski, A., Płaza, M., Łukawski, G., Deniziak, S., Pięta, P., Jasiński, A., Opałka, J., & Węgrzyn, P. (2024). *Supporting Unity developers with an AI-powered asset for multiplayer games development*. Proceedings of the 1st International Workshop on Designing and Building Hybrid Human–AI Systems (SYNERGY 2024). ruj.uj.edu.pl
4. Pingle Studio. (2024). *Multiplayer Game Development Essentials in 2024*. Pingle Studio Blog. Pingle Studio
5. DriversCloud. (2024). *The rise of multiplayer dynamics in 2024 online games*. DriversCloud News. driverscloud.com
6. Unity Technologies. (2024). *2024 Unity Gaming Report – Trends, data & expert tips*. Unity Resources. Unity
7. GeeksforGeeks. (2025). *Object-Oriented Analysis and Design (OOAD)*. Geeks for Geeks Article (updated 12 Jul 2025). GeeksforGeeks
8. ("Object Oriented Analysis & Design Unit Notes", 2024) – *Object Oriented Analysis & Design – OOAD – (CS8592) Notes, Question Papers & Syllabus*. Stucor. stucor.in
9. ResearchGate. (2015). *Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey*. Although earlier than 2024, you might still reference this for background; however you may seek the most recent version. ResearchGate
10. ScienceDirect. (n.d.). *Massively Multi-Player Online Games – An Overview*. While not strictly 2024/2025, it provides foundational context for multiplayer gaming systems. ScienceDirect

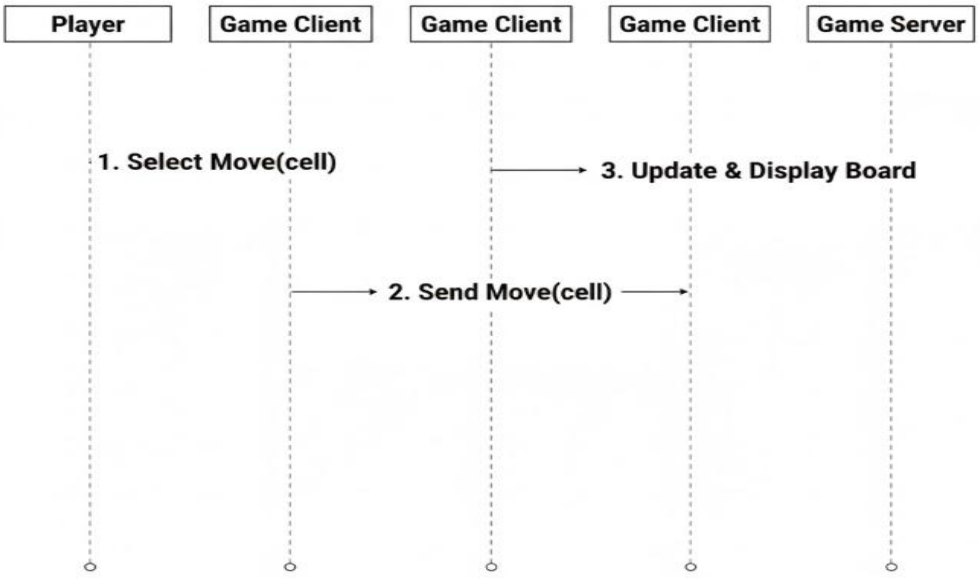
APPENDICES

MODULE 1 USER

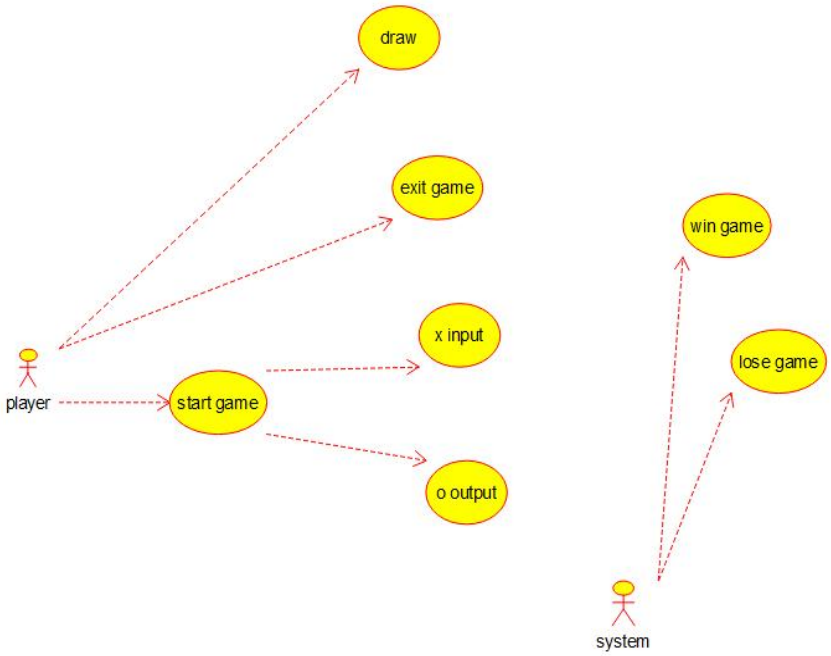
CLASSDIAGRAM



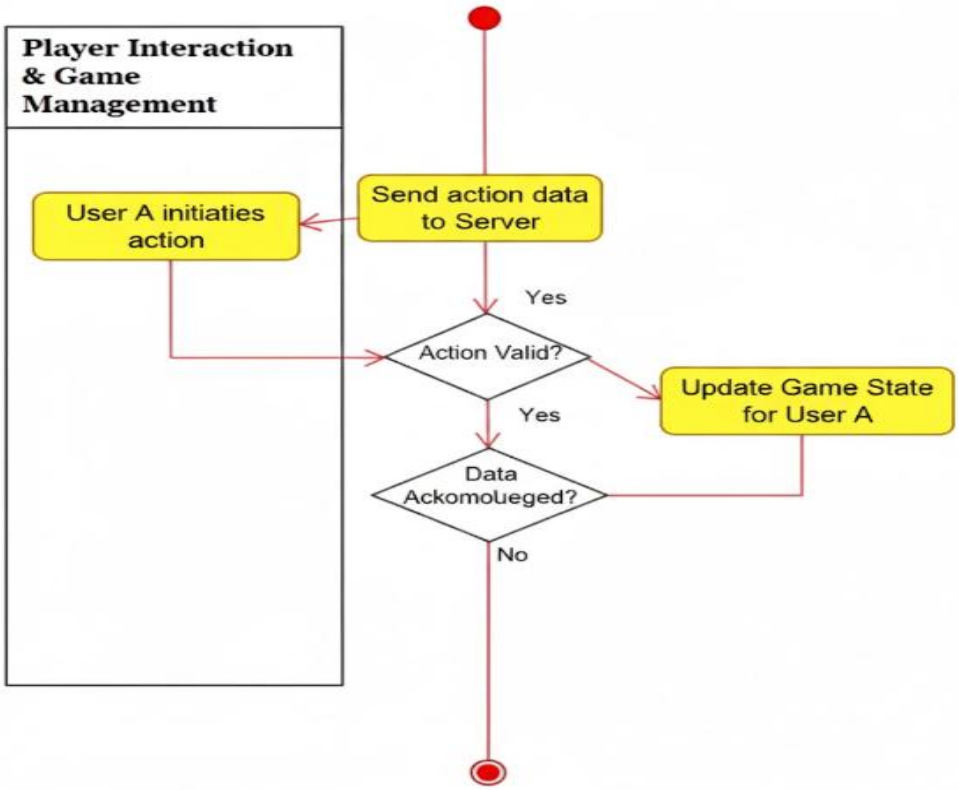
SEQUENCE DIAGRAM



USECASE DIAGRAM

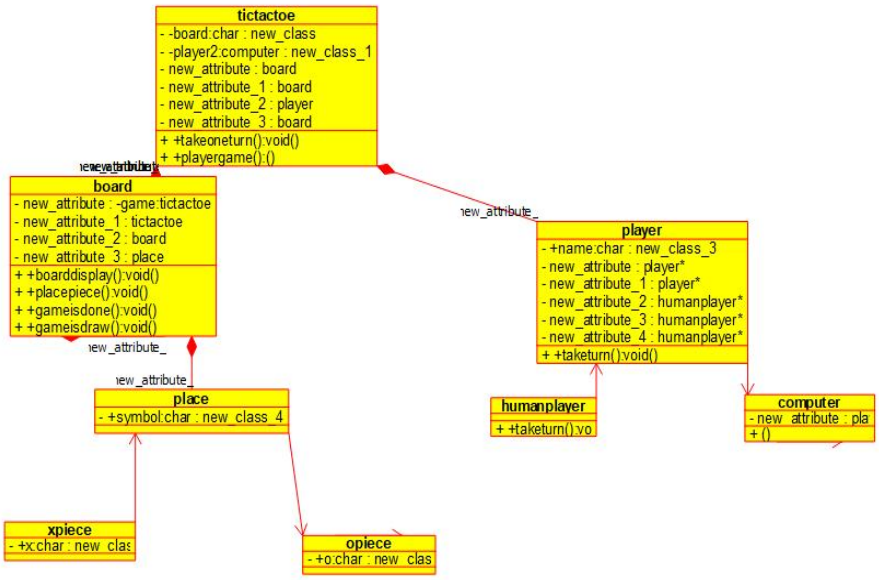


ACTIVITY DIAGRAM

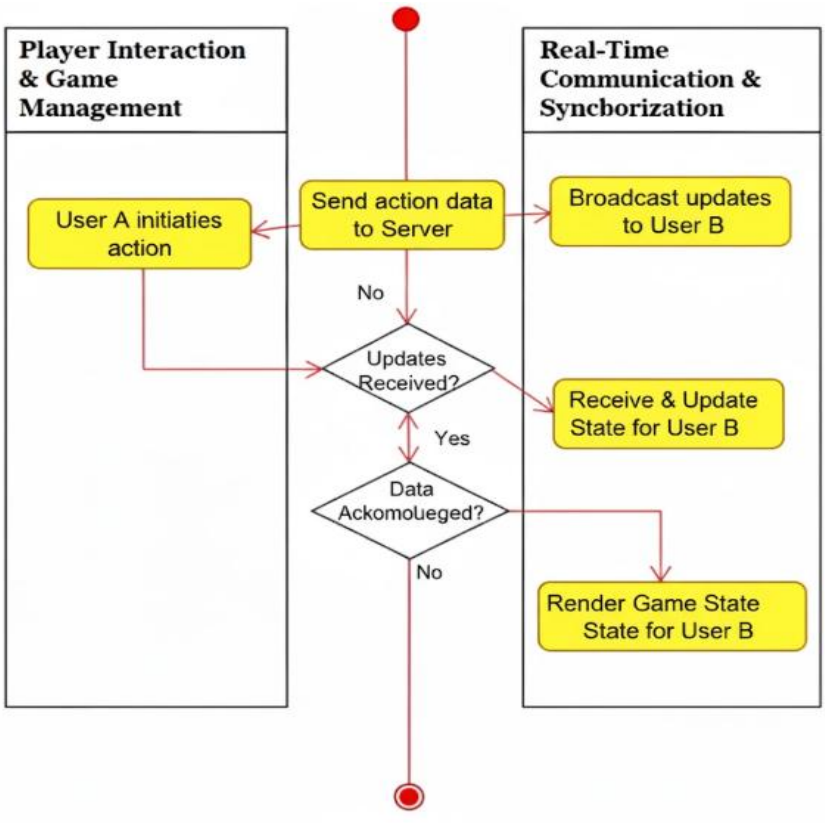


MODULE 2:

CLASS DIAGRAM



ACTIVITY DIAGRAM



OVERALL MODULE: