IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

Multiplayer Matchmaking Engine (Pong Game)

Omkumar Moury[1] , Mayur There[2], Prof. Vrushali Awale[3]
Students, Department of Computer Science[1,2]
Professor, Department of Computer Science[3]
Rajiv Gandhi College of Engineering, Research and Technology, Chandrapur

Abstract: *This project presents a comprehensive multiplayer Pong game engine designed for real-time gaming applications using serverless architecture. By leveraging Java for client-side implementation and Amazon Web Services (AWS) cloud infrastructure including Lambda functions, DynamoDB, and API Gateway, the system processes multiplayer matchmaking and game state management efficiently. The implementation emphasizes scalable architecture, real-time synchronization, and cost-effective serverless deployment. The system's design ensures automatic scaling and adaptability, making it suitable for deployment across various platforms requiring efficient multiplayer gaming capabilities with minimal infrastructure overhead.*

Keywords: Multiplayer Gaming, Serverless Architecture, AWS Lambda, DynamoDB, Game Engine, Real-time Systems, Matchmaking, Java.

I. INTRODUCTION

Multiplayer game development involves creating systems that can handle concurrent players, maintain game state consistency, and provide real-time interaction capabilities. Using Java for client-side development combined with AWS serverless architecture, this can be accomplished by implementing game logic, matchmaking algorithms, and persistent storage solutions. Tools like AWS Lambda, DynamoDB, and API Gateway, along with Java frameworks facilitate this process. Each player action is processed instantly through serverless functions, enabling the application of cloud-based game engines to manage multiplayer sessions and maintain authoritative game state. This technology finds use in various gaming domains, including casual multiplayer games, competitive gaming platforms, and educational game development, serving as a powerful and innovative solution for scalable multiplayer gaming systems.

II. CONTENT DETAILS

A. Aims and Objectives.

1. Real-Time Multiplayer Gaming: Develop a system capable of handling concurrent players in real-time Pong matches.
2. Efficient Matchmaking: Implement algorithms to pair players based on skill rating and geographic proximity.
3. Serverless Architecture: Utilize AWS Lambda functions for scalable, cost-effective backend processing.
4. State Synchronization: Maintain consistent game state across distributed clients.

B. Methodology.

The system follows these core steps:

- Environment Setup: Configure Java development environment and AWS services including Lambda, DynamoDB, and API Gateway.
- Client Development: Create Java-based game client using Swing framework for UI and custom networking components.
- Backend Architecture: Design serverless functions for matchmaking, game state management, and player actions.
- Database Design: Implement DynamoDB schema for player profiles, game sessions, and match history.
- Real-time Communication: Establish WebSocket connections through API Gateway for live game updates.

DOI: 10.48175/568

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

- Game Logic Implementation: Develop collision detection, physics simulation, and scoring systems.
- Testing & Deployment: Conduct performance testing and deploy to AWS cloud infrastructure.

C. Module Description.

This module focuses on multiplayer game engine development using Java and AWS serverless architecture. It covers the fundamentals of game development, cloud computing, and real-time systems with hands-on implementation using Java, AWS Lambda, DynamoDB, and API Gateway. The module addresses client-server architecture, serverless computing concepts, database design, and real-time communication protocols, enabling students to create scalable multiplayer gaming applications.

Key topics include:

- Java Game Development Fundamentals
- AWS Lambda and Serverless Architecture
- DynamoDB Database Design and Operations
- API Gateway and WebSocket Communication
- Real-time Game State Synchronization
- Matchmaking Algorithm Implementation
- Performance Optimization and Cost Analysis
- Cloud Deployment and Monitoring

At the end of this module, students will be able to design and deploy scalable multiplayer game engines using modern cloud technologies for applications such as casual gaming, educational platforms, and competitive gaming systems.

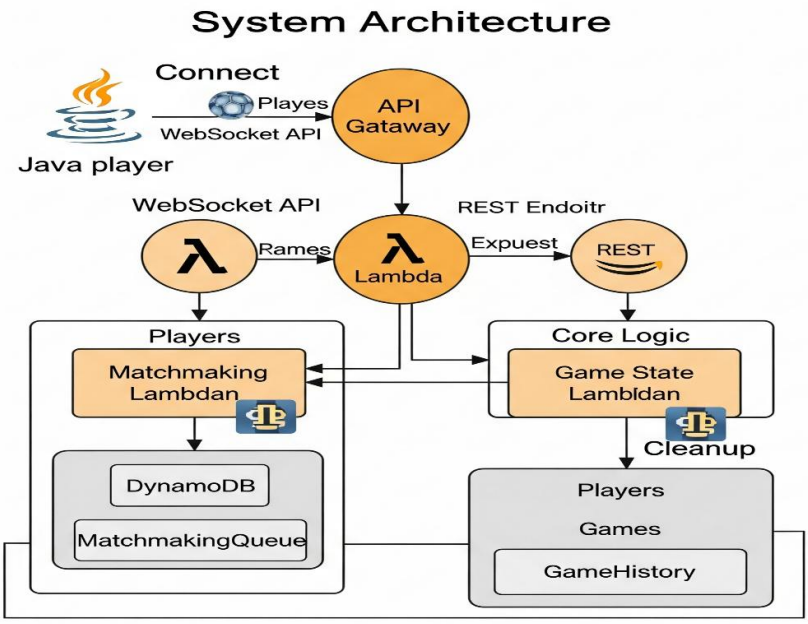
D. Algorithm Description

The game processing pipeline comprises:

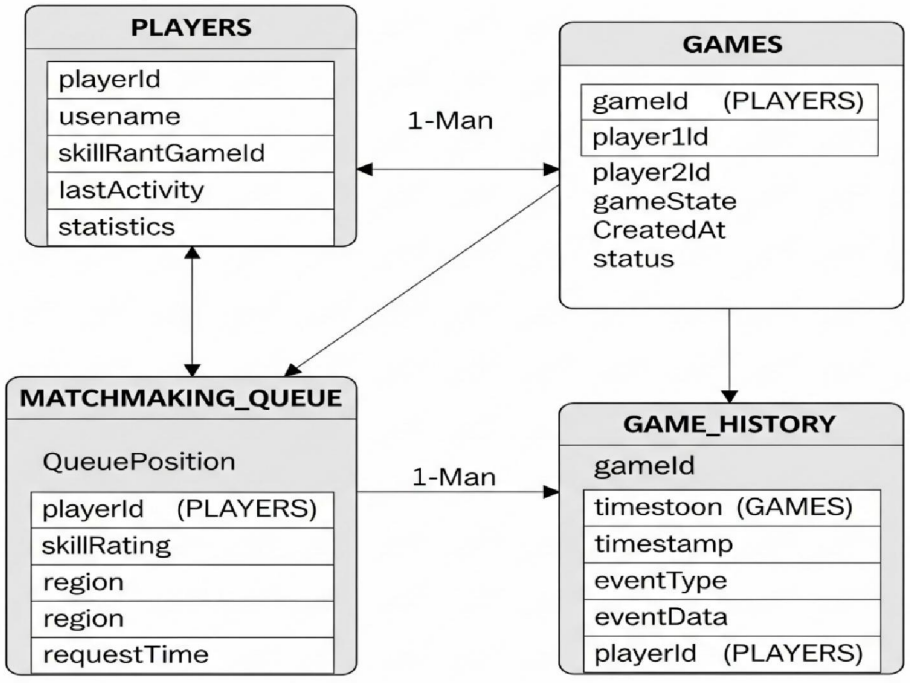
- **Client Initialization**: Start Java application and establish connection to AWS API Gateway.
- **Player Authentication**: Authenticate player credentials and retrieve profile information from DynamoDB.
- **Matchmaking Request**: Submit matchmaking request to Lambda function with player skill rating and preferences.
- **Match Creation**: Lambda function processes queue and creates game session when suitable opponent is found.
- **Game State Setup**: Initialize game state in DynamoDB with player positions, ball location, and game parameters.
- **Real-time Updates**: Process player inputs through WebSocket connections and update authoritative game state.
- **Collision Detection**: Lambda function handles ball-paddle and ball-wall collision detection and physics.
- **Score Management**: Update player scores and broadcast changes to connected clients.
- **Game Completion**: Handle game end conditions and update player statistics and match history.
- **Session Cleanup**: Remove expired game sessions and release resources.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

E. UML & ER Diagram



SYSTEM ARCHITECTURE DIAGRAM

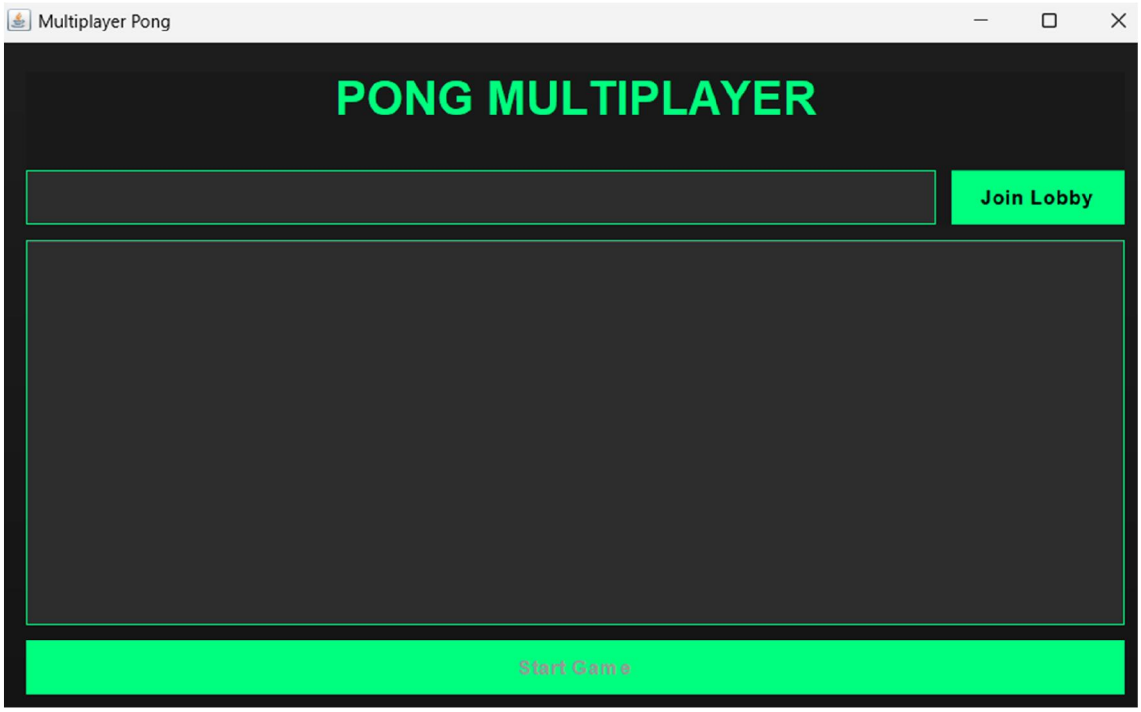


DOI: 10.48175/568

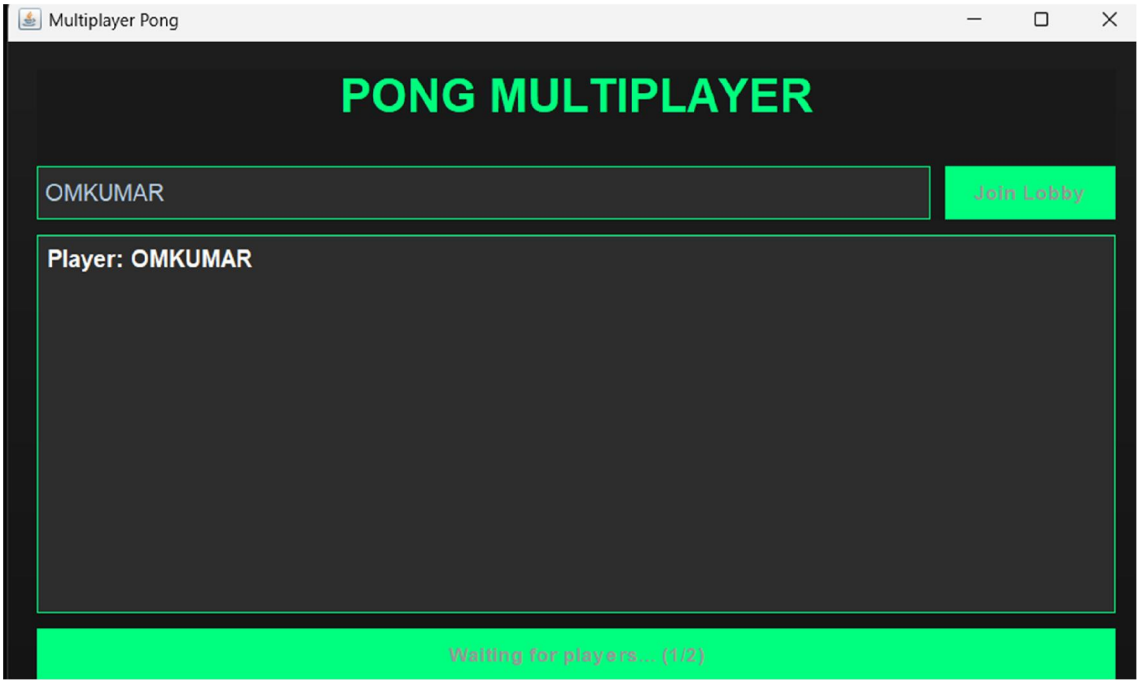
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

III. OUTPUT

Game lobby

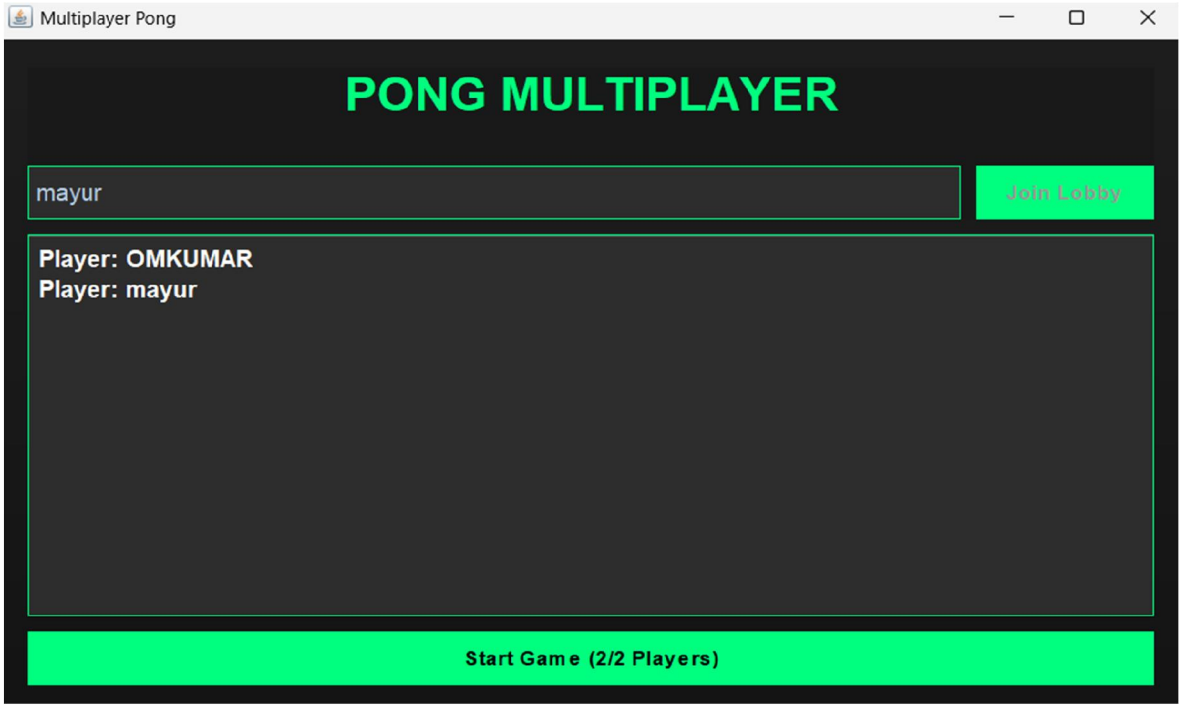


After connecting one player

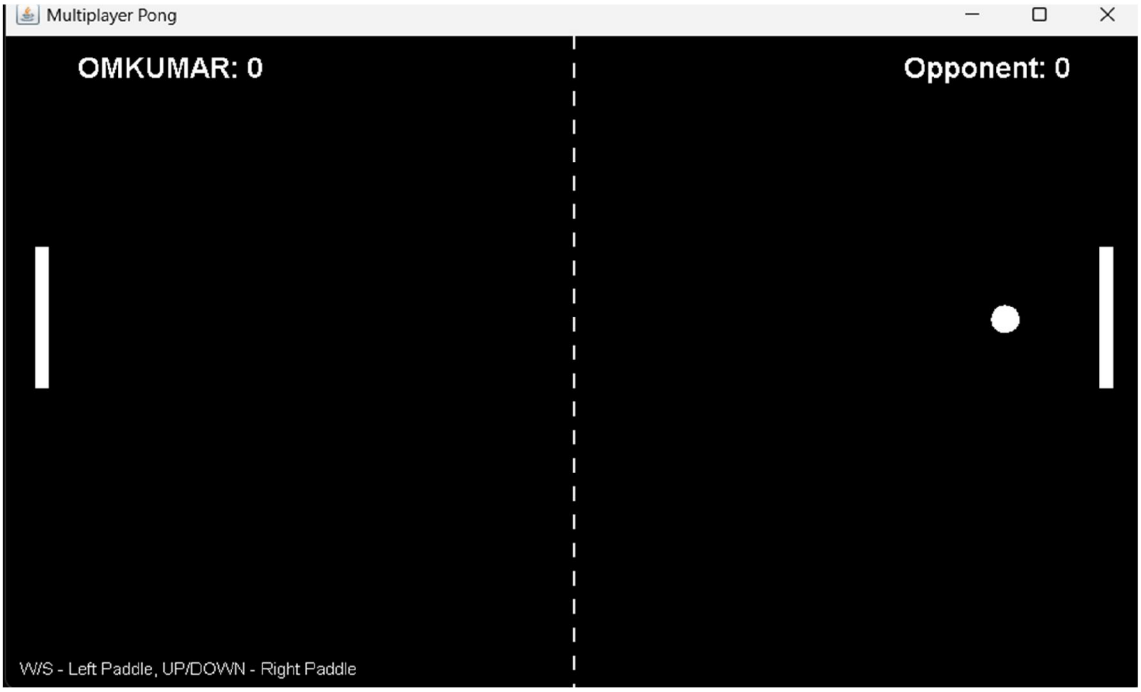


IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

After connecting 2 palyer



After click on start game (Game started)



DOI: 10.48175/568

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 8, June 2025
Impact Factor: 7.67

Console

```
Main started
MainController constructor called
WebSocketClient constructor called
Attempting to connect to WebSocket: wss://0yg6rgepmk.execute-api.us-east-1.amazonaws.com/prod/
Connected to WebSocket!
WebSocket opened
Sending join message: {"name":"mayur","action":"joinLobby"}
Sending message: {"name":"mayur","action":"joinLobby"}
Raw message received: {"type": "lobbyUpdate", "players": ["OMKUMAR", "mayur"]}
MainController processing message: {"type": "lobbyUpdate", "players": ["OMKUMAR", "mayur"]}
Processing lobby update with players: OMKUMAR, mayur
Updating lobby UI with 2 players
LoginLobbyPanel updating with 2 players
Setting start button enabled: true
Raw message received: {"type": "gameStarted", "gameStarted": true}
MainController processing message: {"type": "gameStarted", "gameStarted": true}
Game start message received - switching to game panel
Switching to game panel...
```

IV. CONCLUSION

The developed multiplayer Pong game engine demonstrates the practical application of serverless architecture in real-time gaming applications. By integrating AWS Lambda, DynamoDB, and API Gateway with Java client development, the system achieves scalable, cost-effective multiplayer gaming capabilities with automatic scaling and minimal operational overhead. The modular serverless architecture enables efficient matchmaking, real-time state synchronization, and persistent game data management. Its cloud-native design supports global scalability, making it suitable for applications in casual gaming, educational platforms, and competitive gaming environments. The use of managed AWS services allows for reduced infrastructure complexity and pay-per-use cost optimization. Future enhancements may involve implementing advanced matchmaking algorithms using machine learning, deploying edge computing solutions for reduced latency, and integrating additional game modes and features. Additionally, incorporating mobile client support, spectator functionality, and tournament management systems can further expand the platform's capabilities and user engagement in practical gaming scenarios.

V. AKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who supported me throughout the course of this project. First and foremost, I am deeply thankful to our mentor, Prof. [Guide Name], for his invaluable guidance, encouragement, and continuous support during the course of this work. His insights and expertise in cloud computing and game development were instrumental in the successful completion of this project. I also extend my heartfelt thanks to the Department of Computer Science, [College Name], [City], India, for providing the necessary resources and a conducive environment for learning and development. Special appreciation goes to Amazon Web Services for providing the cloud infrastructure and documentation that made this serverless implementation possible.

REFERENCES

[1]. Oracle Corporation, "Java Platform, Standard Edition Documentation," Oracle, 2023.
[2]. Amazon Web Services, "AWS Lambda Developer Guide," AWS Documentation, 2023.
[3]. Amazon Web Services, "Amazon DynamoDB Developer Guide," AWS Documentation, 2023.
[4]. Amazon Web Services, "Amazon API Gateway Developer Guide," AWS Documentation, 2023.
[5]. https://www.youtube.com
[6]. https://claude.ai/new (claude AI)
[7]. https://chatgpt.com/ (CHAT GPT)