Software Design Document

# Executive Summary

The client, The Gaming Room, requested the design of a multi-platform application for their game, Draw It or Lose It. Originally developed for Android, the game is a team-based drawing and guessing competition, similar to Win, Lose or Draw. The goal of this project was to design a scalable, distributed application that supports multiple operating systems, ensures security, manages concurrent gameplay, and provides efficient user and team management.

# System Architecture & Design Constraints

The system follows a client-server architecture with RESTful APIs for communication. Design constraints include budget limitations, cross-platform compatibility, and security considerations. The design must allow multiple simultaneous game sessions and ensure unique identifiers for games, teams, and players.

# Domain Model UML

The domain model includes the following primary entities:  
- GameService (Singleton)  
- Game  
- Team  
- Player  
  
The GameService acts as the system controller, managing multiple game instances. Each game is associated with teams and players, and identifiers ensure uniqueness across sessions.

\*(UML Diagram should be included here as an image in the final submission.)\*

# Platform Evaluation

The application was evaluated for deployment across Linux, Windows, Mac, and Mobile platforms. Considerations included licensing costs, available development tools, scalability, and ease of deployment.  
  
- Linux: Cost-effective, scalable, widely used for server hosting.  
- Windows: Strong support for development tools but higher licensing costs.  
- Mac: Useful for iOS development but less common for server hosting.  
- Mobile (Android/iOS): Provides user accessibility but introduces device-specific constraints.  
  
Based on this evaluation, Linux is recommended for server hosting due to its cost efficiency and scalability.

# Design Patterns

The design makes use of the following design patterns:  
- Singleton: Applied to the GameService class to ensure a single point of control for managing games.  
- Iterator: Used for iterating over collections of players and teams efficiently.  
  
These patterns improve code organization, scalability, and maintainability.

# Memory and Storage Management

The system leverages memory management techniques to handle multiple concurrent game sessions. Object references are managed using garbage collection, and database storage ensures persistence of game states. Caching strategies are recommended for frequently accessed data, such as active games and player sessions.

# Distributed Systems & Networking

The system is designed as a distributed application using a client-server model. The server provides RESTful APIs that clients on various platforms can consume. Basic Authentication is implemented for security, with the option to integrate more advanced mechanisms in the future. Network communication is secured via HTTPS.