Draw It or Lose It

CS 230 Project Software Design

Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	2025-03-23	Braxton Schumann	Initial revision. Add client details, requirements, and constraints
2.0	2025-04-06	Braxton Schumann	Add Evaluation
3.0	2025-04-20	Braxton Schumann	Add Recommendations

Executive Summary

The Gaming Room aims to expand its game, Draw It or Lose It, from an Android-only application to a web-based, cross-platform solution. This transition will enable users to play from multiple devices while maintaining a seamless and engaging gaming experience. To achieve this, the game must be designed as a scalable and efficient distributed system that supports multiple teams and players while enforcing unique names for games and teams. Additionally, only one instance of the game should exist in memory at any given time, requiring the implementation of a singleton pattern for game management.

This document outlines the design constraints and key considerations necessary for developing a robust and maintainable web-based gaming application. By leveraging industry-standard design patterns and best practices, the system will ensure optimal performance, data integrity, and a smooth user experience.

Requirements

The Gaming Room's game application must meet several business and technical requirements to ensure functionality, scalability, and security.

From a **business** perspective, the game must support multiple teams, with multiple players per team, and enforce unique game and team names to prevent duplication. Users should be able to check whether a game or team name is already in use before proceeding. The gameplay must align with the original Android version's structure, including four rounds with timed stock image rendering. Additionally, the system should be accessible across multiple platforms, including desktop and mobile devices.

From a **technical** perspective, the game service must enforce a singleton pattern to ensure that only one instance of a game exists in memory at a given time. The system must also utilize an iterator pattern to check for duplicate game and team names. Since the game will operate in a web-based environment, it must support multiple concurrent game sessions without performance degradation. Proper database integration is essential for storing and retrieving player, team, and game data efficiently. Security measures such as authentication and input validation must be in place to prevent unauthorized access and maintain fair gameplay.

Design Constraints

Developing Draw It or Lose It as a web-based distributed application introduces several constraints that must be addressed to ensure **efficiency**, **scalability**, and **security**.

The **web-based environment** presents challenges such as cross-browser compatibility, network latency, and real-time communication requirements. The game must function correctly across different browsers while minimizing lag caused by network limitations. Implementing WebSockets or similar technologies will help maintain low-latency interactions for a smooth multiplayer experience.

Scalability and **performance** are critical considerations, as the system must handle multiple users simultaneously. Efficient server load management, database indexing, and optimized session management are necessary to prevent performance bottlenecks. The game engine must support active sessions without excessive memory consumption, while ensuring seamless user interactions. From an application logic standpoint, enforcing the singleton pattern ensures that only one game instance exists in memory, preventing conflicts and redundancy. Additionally, the iterator pattern will be implemented to check for duplicate game and team names before allowing new instances to be created. These patterns will streamline the game management process and prevent unintended duplication.

Security is a major data integrity concern, requiring authentication, authorization, and input validation. The system should implement secure login mechanisms, such as OAuth or token-based authentication, to prevent unauthorized access. Proper validation must be in place to reject invalid or malicious inputs when users create game sessions, teams, or player names. Additionally, session security measures should be enforced to protect against potential attacks such as session hijacking.

Finally, **user experience (UX)** constraints must be addressed to ensure a responsive and intuitive interface. The UI should be mobile-friendly and adapt to different screen sizes while maintaining functionality. Minimal latency is essential, as real-time gameplay depends on quick rendering and seamless player interactions. The system must be designed with efficiency in mind to provide a smooth gaming experience across all supported platforms.

System Architecture View

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

Domain Model

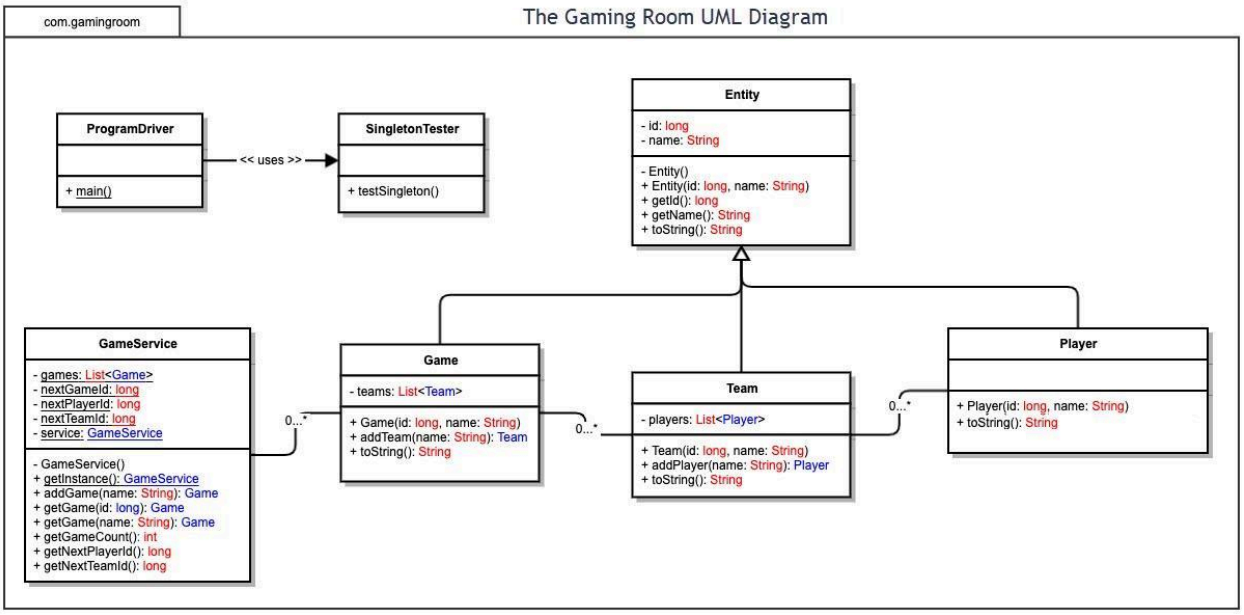
The UML class diagram for Draw It or Lose It demonstrates a structured approach using inheritance, encapsulation, and the singleton pattern to ensure efficient game management. At the core, the Entity class serves as a base for Game, Team, and Player, allowing them to inherit common attributes (id and name) and methods (getId(), getName(), toString()). This promotes code reuse and consistency across related classes.

The GameService class follows the singleton pattern, ensuring only one instance exists to manage game-related operations. It maintains a list of games and generates unique identifiers (nextGameId, nextPlayerId, nextTeamId) to prevent duplicate instances. The hierarchical relationships ensure structured management—each Game can have multiple Teams, and each Team consists of multiple Players. This

one-to-many aggregation effectively organizes game elements while maintaining independence between instances.

Encapsulation protects data by keeping attributes private, while inheritance allows shared behavior, reducing redundancy. The singleton pattern in GameService optimizes memory usage and ensures centralized game management. Abstraction allows GameService to handle essential game operations without exposing unnecessary details.

This design meets The Gaming Room's requirements by enforcing unique names, maintaining a single game instance, and ensuring scalable, structured development. It provides a solid foundation for future enhancements while keeping the system organized and efficient.



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Mac is not a common choice for hosting web-based applications due to limited enterprise-grade server support. While macOS is Unix-based and provides stability, security, and strong development tools, it lacks scalability and enterprise-level support compared to Linux or Windows Server. Hosting options are also limited, making Mac less ideal for large-scale deployment.	Linux is the most widely used operating system for hosting web applications due to its stability, security, scalability, and open-source nature. It supports various web servers such as Apache, Nginx, and Tomcat, making it cost-effective and highly customizable. Many cloud providers favor Linux-based servers due to their efficiency.	Windows Server provides a reliable and widely used environment for hosting web applications, especially for enterprises using Microsoft technologies such as ASP.NET, IIS (Internet Information Services), and SQL Server. However, licensing costs can be high, and Windows-based servers may require more resources for maintenance and security updates.	Mobile applications typically rely on backend APIs hosted on web servers. These servers are usually Linux-based due to cost-effectiveness and reliability, although Windows Server can also be used. Mobile-friendly backend solutions like Firebase, AWS, and Azure are commonly integrated.

Client Side	macOS supports powerful development tools such as Xcode (for Apple applications), Eclipse, and JetBrains IDEs for cross-platform development. Programming languages like Swift, Java, Python, and JavaScript are commonly used, making it suitable for developing both native and web-based applications.	Linux-based clients are less common, but web applications can still support Linux users through browsers like Firefox and Chrome. However, ensuring broad compatibility with Linux distributions can require extra testing and development time.	Windows dominates the desktop market, ensuring a broad user base. Supporting Windows clients means optimizing for Microsoft Edge, Chrome, and Firefox, and ensuring compatibility with various hardware configurations. Windows' vast developer ecosystem allows easier cross-platform development and testing.	Mobile development requires separate considerations for iOS (Swift) and Android (Kotlin/Java). Cross-platform frameworks like Flutter, React Native, and Xamarin allow simultaneous development for both platforms, reducing cost and time. Ensuring performance and UI consistency across different screen sizes and operating systems is a key challenge.

Development Tools	macOS supports powerful development tools such as Xcode (for Apple applications), Eclipse, and JetBrains IDEs for cross-platform development. Programming languages like Swift, Java, Python, and JavaScript are commonly used, making it suitable for developing both native and web-based applications.	Linux supports a variety of open-source tools, including Eclipse, IntelliJ IDEA, VS Code, and terminal-based tools for efficient development. Programming languages such as Java, Python, JavaScript, and PHP are widely supported, making Linux an excellent choice for full-stack development.	Windows provides access to a wide range of IDEs such as Visual Studio, Eclipse, IntelliJ IDEA, and NetBeans. Popular programming languages for web development include C#, Java, JavaScript, and Python, making Windows a versatile platform for both client and server-side development.	Tools: iOS development primarily uses Xcode and Swift, while Android development relies on Android Studio and Kotlin/Java. Cross-platform tools such as Flutter, React Native, and Unity allow developers to build games and applications that run on both iOS and Android with shared codebases.

Recommendations

1. **Operating Platform:**
   For expanding *Draw It or Lose It* across multiple environments, Linux is the recommended operating platform for the server side. Linux offers stability, strong security features, scalability, and cost-effectiveness, all of which are essential for supporting a web-based application with real-time features. For the client side, a platform-independent strategy will be used, utilizing responsive web applications and cross-platform mobile apps to reach Windows, macOS, iOS, and Android users seamlessly.

2. **Operating Systems Architectures:**
   The server will use a 64-bit Linux architecture and follow a microservices design pattern. This architecture splits the application into smaller, independent services—such as user management, game sessions, and scoring systems—that communicate through lightweight APIs. This ensures high scalability, fault isolation, and ease of updates without disrupting the entire system. Containerization technologies like Docker and orchestration tools like Kubernetes will manage deployment, further enhancing flexibility and fault tolerance.

3. **Storage Management:**
   A cloud-based database system is recommended for storing application data. MongoDB, a NoSQL database, will be used for storing user profiles, game sessions, and team data due to its flexibility and ability to scale horizontally. For large asset storage, such as drawing images or media files, a cloud object storage solution like Amazon S3 will be utilized. This combination allows for efficient, secure, and scalable data management to support fluctuating user demands.

4. **Memory Management:**
   The Linux platform efficiently manages memory through virtual memory, paging, and segmentation. These techniques dynamically allocate memory to active processes, preventing memory leaks and optimizing performance. To further enhance performance, in-memory caching solutions like Redis will store frequently accessed game data, minimizing database calls and reducing latency during gameplay.

5. **Distributed Systems and Networks:**
   The game will operate as a distributed system to allow seamless communication across platforms. RESTful APIs will handle standard operations, while WebSockets will manage real-time game interactions. A global content delivery network (CDN) will distribute static assets, and load balancers will manage traffic efficiently. Redundancy, failover strategies, and automated backups will be implemented to maintain service availability during network interruptions or server outages, ensuring minimal downtime and a smooth user experience.

6. **Security:**
   Security measures will include using SSL/TLS protocols for encrypted data transmission, securing user authentication with OAuth 2.0, and issuing tokens via JWT for session management. Sensitive data will be encrypted at rest and in transit.

Role-based access control (RBAC) will ensure only authorized users access critical functions. Regular security audits, DDoS mitigation strategies, and secure coding practices will further protect user information and ensure compliance with data protection standards across all supported platforms.