THE GAMING ROOM

Draw It or Lose It
CS 230 Project Software Design Template
Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	05/17/25	Connor Martin	Initial version of the software design document for the web-based version of Draw It or Lose It, including the executive summary, design constraints, domain model analysis, evaluation of operating platforms, and recommendations.
2.0	06/01/25	Connor Martin	Added platform evaluation table and summary for Project Two. Rewrote the Evaluation section introduction and Domain Model to reflect a high-level design focus. Ensured the document aligns with software design documentation best practices and my instructors feedback.
3.0	6/14/25	Connor Martin	Finalized software design document for portfolio submission. Added reflective content for the CS 230 Module Eight Journal, cleaned up formatting and language for clarity, and confirmed inclusion of all project requirements. Updated README in GitHub repository to include project summary and personal reflection in response to course journal prompts.

Instructions

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

Executive Summary

The Gaming Room is seeking to expand its existing Android-based game, Draw It or Lose It, to a web-based platform that can serve multiple platforms. The objective is to create a streamlined game application that allows for multiple teams and players while ensuring the uniqueness of game and team names. Additionally, the client requires a single instance of the game to exist in memory, leveraging software design patterns such as Singleton and Iterator to manage game instances effectively. The proposed solution involves developing a game application in Java using object-oriented principles, adhering to the specified software requirements, and optimizing the application's architecture to support scalability and cross-platform functionality.

Requirements

1. **Business Requirements:**

- The Gaming Room wants to expand their game, "Draw It or Lose It," from an Android-only app to a web-based version that can be accessed on multiple platforms.
- The game must support multiple teams and multiple players per team, with each game and team name being unique.
- The game flow consists of four rounds with each round lasting one minute. If the team doesn't guess the puzzle in time, the remaining teams have one guess each with a 15-second time limit.

2. **Technical Requirements:**

- Implement a Singleton pattern to make sure there is only one instance of the game service running in memory at any time.
- Use an Iterator pattern to manage game, team, and player objects effectively and enforce name uniqueness.
- Create a base `Entity` class to hold common attributes like ID and name. The `Game`, `Team`, and `Player` classes will inherit from this class.
- Develop methods for adding and retrieving games, teams, and players while ensuring unique identifiers and names.
- The application must be written in Java using object-oriented programming principles and must follow standard coding best practices.
- Include error handling to prevent duplicate names and maintain data integrity.

Design Constraints

The development of the web-based version of "Draw It or Lose It" involves several design constraints that must be addressed to ensure the application functions effectively in a distributed web environment.

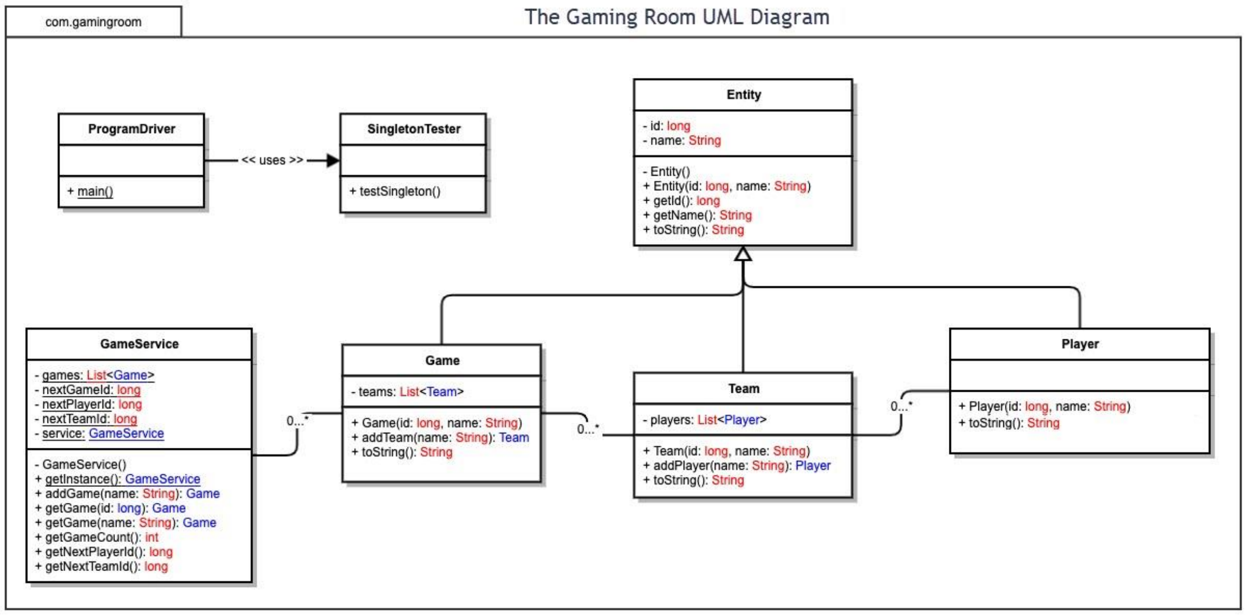
1. **Single Instance Limitation:**
    - The application must implement a Singleton pattern to ensure that only one instance of the game service is running at any given time. This prevents conflicting game states and ensures consistent gameplay across all sessions.
2. **Name Uniqueness:**
    - Both game and team names must be unique to avoid confusion and conflicts during gameplay. The application must use the Iterator pattern to check for existing names before adding new games, teams, or players.
3. **Distributed System Considerations:**
    - The game will operate in a web-based environment, meaning that data synchronization between clients and the server is essential. Network latency and potential connectivity issues must be considered, particularly during critical moments like the 15-second guessing phase after each round.
4. **Memory Management:**
    - Memory must be efficiently managed to handle multiple games and teams without overwhelming system resources. The use of unique identifiers and effective garbage collection is necessary to maintain optimal performance.
5. **Cross-Platform Compatibility:**
    - Since the application will be accessible on multiple platforms, the design must account for differences in operating systems, browsers, and screen sizes. This includes ensuring that the application logic functions consistently regardless of the client’s platform.

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 domain model for *Draw It or Lose It* defines the primary entities involved in the game and the relationships between them. The system is structured to support multiple games, each containing one or more teams, and each team consisting of multiple players. All core entities—Game, Team, and Player—share common attributes such as unique identifiers and names, which promotes consistency and data integrity throughout the application.
The model uses inheritance to define these shared properties, helping to reduce code duplication and simplify data handling. It also employs design patterns like Singleton to manage game instances and ensure only one service instance controls game flow at a time. Iterator-like behavior is used in validating name uniqueness and navigating through collections of games, teams, and players.
This structure supports scalability, modularity, and maintainability, which are essential for a distributed, web-based gaming environment. It also prepares the application for cross-platform deployment by enforcing consistent logic and data access across all client types.



Evaluation

The following evaluation examines five platforms—Linux, macOS, Windows, Android, and iOS—based on their capabilities to support the game *Draw It or Lose It* in a distributed, cross-platform environment. The analysis addresses server-side deployment, licensing, client compatibility, development requirements, and tool availability to help The Gaming Room make an informed decision.

Platform	**Server-Side Capabilities**	**Licensing Costs**	**Client-Side Compatibility**	**Development Considerations**	**Development Tools & Costs**
Linux	Industry standard for scalable web applications. Supports Apache, Nginx, and Docker. Reliable and efficient for high-traffic environments.	Free and open-source. No licensing fees required.	Fully supports modern browsers like Chrome and Firefox. Web app runs smoothly with responsive design.	Requires familiarity with shell scripting and server config. Strong community support and low resource footprint.	Tools: VS Code, Git, Docker, Node.js – all free. No IDE license required.
Windows	Supports IIS and Azure for scalable deployment. Integrates well with enterprise tools. Heavier than Linux but widely used.	Windows Server licenses required. Azure adds potential ongoing subscription costs.	Compatible with Chrome, Edge, Firefox. Full web app functionality achievable.	Developers need knowledge of IIS, .NET, and Windows environments. Slightly more complex setup.	Tools: Visual Studio (Community edition is free), PowerShell, SQL Server Express. Pro versions may have costs.
macOS	Possible to host using Apache/nginx but not commonly used for production. More often used for testing or frontend dev.	High hardware cost. No separate server license but macOS must run on Apple hardware.	Fully compatible with Safari, Chrome, Firefox. Runs responsive web apps reliably.	Best used for iOS/browser testing rather than server deployment. Limited scalability.	Tools: Xcode, Terminal, Homebrew. Most are free, but hardware cost is high.
Android	Not designed for hosting. Mobile OS used strictly as a client.	Free SDK and emulator tools.	Supports web apps in mobile Chrome and WebView. Requires responsive/mobile-first layout.	Developers must test across devices, screen sizes, and OS versions. Touch input is primary interface.	Tools: Android Studio, Java/Kotlin, HTML/JS – all free.
iOS	Not usable for server hosting. Used only as a mobile client platform.	Requires Apple Developer account ($99/year).	Supports responsive web apps via Safari. HTML5, CSS3, and JS needed for compatibility.	Must comply with Apple's UX standards and screen scaling. Testing required on physical iOS devices.	Tools: Xcode (free), Swift, Safari dev tools. Dev license is the main cost.

Recommendations

The following recommendations address the technical components needed to successfully expand *Draw It or Lose It* to a distributed, cross-platform environment. These suggestions take into account system architecture, memory and storage needs, communication across networks, and essential security protections.

1. Operating Platform

The recommended operating platform for expanding *Draw It or Lose It* is a Linux-based server environment, specifically Ubuntu Server. Linux is stable, cost-effective, and widely supported in enterprise-level hosting. It integrates well with Java and offers reliable tools for building scalable server applications. It is also open-source, meaning there are no licensing fees, which helps reduce long-term costs for The Gaming Room.

2. Operating Systems Architectures

Linux operates using a monolithic kernel architecture, where core system functions like process scheduling, memory management, and device drivers operate in kernel space for improved performance. This architecture also supports modularity, meaning components like drivers can be loaded or unloaded as needed without rebooting. These characteristics make Linux highly customizable and efficient—ideal for supporting a centralized game server while maintaining compatibility with client platforms like Windows, macOS, Android, and iOS through the use of Java.

3. Storage Management

To manage data effectively, I recommend using the ext4 file system combined with Logical Volume Management (LVM). This allows for dynamic resizing and flexible disk partitioning, making it easier to scale as the game grows. For managing game, player, and team data, MySQL is a strong option due to its seamless integration with Java, high performance, and widespread use. MySQL also provides transaction handling and supports structured data, which is useful for keeping game instances and records organized and secure.

4. Memory Management

Linux uses virtual memory, demand paging, and shared memory to optimize RAM usage. These features will allow the game server to manage multiple sessions at once while maintaining performance. The Singleton pattern will also reduce memory overhead by ensuring only one instance of the GameService is ever active. These strategies combined will prevent unnecessary duplication of resources and support stable performance even during high traffic.

5. Distributed Systems and Networks

Since *Draw It or Lose It* needs to support real-time gameplay across multiple devices, a distributed system architecture is essential. I recommend using a RESTful API for client-server communication and considering WebSockets for real-time events like drawing updates or time countdowns. These technologies are widely supported and allow smooth communication over HTTP/HTTPS. To handle latency and connectivity issues, reconnect logic and error handling should be implemented. Deploying the server in a cloud environment with load balancing and failover support would also help ensure uptime and reliability.

6. Security

Protecting user information and maintaining secure gameplay is critical. All communication between clients and the server should be encrypted using TLS. On the server side, Linux offers strong tools for user authentication, firewall management, and file access controls. User credentials should be securely hashed and salted using algorithms like bcrypt. Additionally, the application should implement role-based access control (RBAC) to limit what users can see or do depending on their roles. These protections, along with regular patching and monitoring, will help safeguard against vulnerabilities and maintain trust with users.