THE GAMING ROOM

Draw It or Lose It

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

## Document Revision History

| Version | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 09/16/2024 | Trevor Brandt | Added Executive Summary, Design Constraints, and Domain Model |
| 2.0 | 09/30/2024 | Trevor Brandt | Added Evaluation |
| 3.0 | 10/14/2024 | Trevor Brandt | Added Recommendations |

**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 seeks to transition their existing Android game, "Draw It or Lose It," to a web-based platform accessible across multiple devices. The primary challenges include supporting multiple teams, ensuring unique game and team names, and managing a single active instance of the game.

To address these challenges, we propose implementing a software solution with a robust Game Management System and a Singleton Service. The Game Management System will handle team and player management, while the Singleton Service will ensure only one instance of the game is active at any time, with unique identifiers for games, teams, and players. This approach will ensure a seamless, scalable, and unique gaming experience across platforms.

## Requirements

1. **Multi-Team Support**: The game must accommodate multiple teams participating simultaneously, allowing for a competitive and engaging experience.
2. **Team Composition**: Each team should consist of multiple players, enabling diverse team strategies and interactions.
3. **Unique Game and Team Names**: Game and team names must be unique to prevent confusion and ensure distinct identification within the system.
4. **Single Instance Management**: Only one instance of the game should be active in memory at any time, utilizing unique identifiers to manage games, teams, and players effectively.

## Design Constraints

1. **Scalability:** The web-based application must handle varying loads, including multiple simultaneous users and teams. The design must include scalable server architecture and efficient data handling to maintain performance and responsiveness as user demand fluctuates.
2. **Cross-Platform Compatibility**: The game must be accessible across different web browsers and devices. Development should adhere to web standards and ensure compatibility with major browsers and devices, which may require additional testing and adjustments.
3. **Security:** The application must protect user data and ensure secure interactions between clients and the server. Robust security measures, such as encryption and secure authentication protocols, to safeguard sensitive information and prevent unauthorized access, must be implemented.
4. **Latency and Performance:** The game needs to operate with minimal latency to provide a real-time experience for users. We will optimize server responses and data transmission to reduce delays and ensure a smooth, interactive gaming experience.
5. **User Experience:** The interface must be intuitive and user-friendly across various platforms. A responsive and accessible user interface, ensuring that game controls and interactions are easy to use on different screen sizes and input methods, is paramount in this project.
6. **Data Integrity:** Ensuring the accuracy and consistency of game data, including unique identifiers for games, teams, and players. We'll implement thorough validation and synchronization mechanisms to maintain data integrity across the distributed environment.
7. **Maintenance and Upgrades:** The application should support ongoing maintenance and updates without disrupting user experience. The system will be designed with modular components and deploy updates in a way that minimizes downtime and user impact.

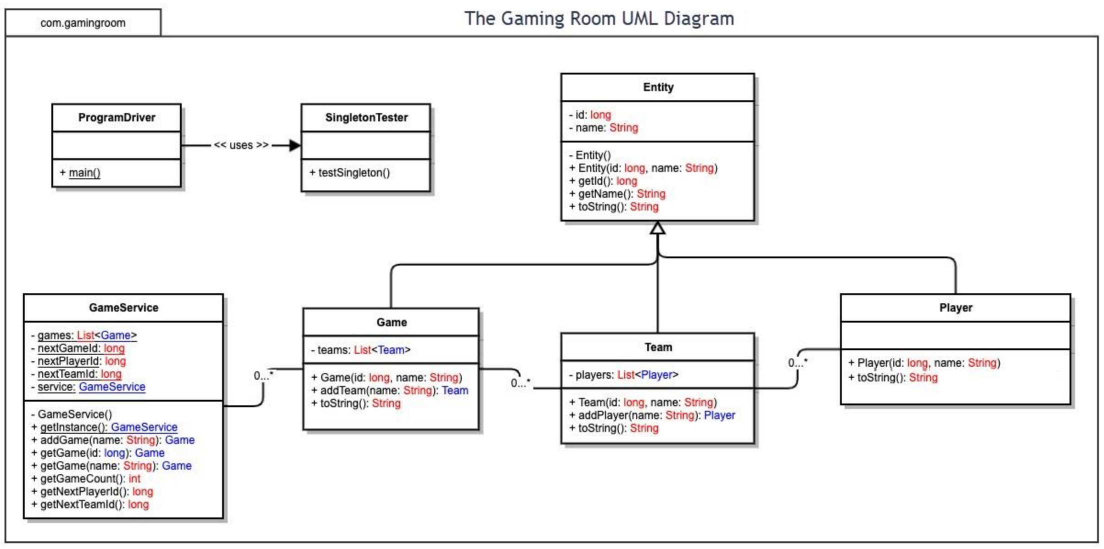
## System Architecture View

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 illustrates the structure and relationships of various classes involved in the game application. At the top level, the ProgramDriver class, with its main() method, uses the SingletonTester class to test singleton behavior. The Entity class serves as a base class with attributes id and name, and methods such as getId(), getName(), and toString(). This class is extended by Game, Team, and Player, demonstrating inheritance, which allows these classes to share common attributes and methods.

The GameService class, implementing the Singleton pattern, manages multiple instances of Game and ensures a single instance of the service exists. It maintains lists of games and unique identifiers for players and teams. The Game class, in turn, contains a list of Team instances, while each Team class holds a list of Player objects, illustrating composition and aggregation. This design adheres to the object-oriented principles of encapsulation by hiding data within each class and providing public methods for interaction. The composition relationships ensure that Game, Team, and Player are structured in a "has-a" manner, allowing complex functionality to be built from simpler components. The Singleton pattern in GameService ensures consistent management of game instances, contributing to data integrity and efficient application management.



## Evaluation

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| Development Requirements | Mac | Linux | Windows | Mobile Devices |
|---|---|---|---|---|
| **Server Side** | Mac systems are built on a Unix-based foundation, which provides stability and security for hosting web-based applications. They support server-based deployment using macOS Server, but this is less common compared to Linux-based solutions due to limited enterprise usage. Licensing costs for macOS are generally higher than Linux but lower than Windows. However, macOS servers are not as widely used for large-scale web hosting, making it less suitable for highly scalable web applications. | Linux is a popular choice for hosting web-based applications due to its stability, security, and open-source nature, which eliminates licensing costs. It offers a wide range of server-based deployment methods, including support for Apache, Nginx, and other web servers, making it highly scalable and flexible. Linux is well-suited for high-performance, large-scale applications, but it may require a more experienced IT team due to its command-line-based management and configuration tools. | Windows provides robust support for hosting web-based applications through IIS (Internet Information Services) and integrates well with Microsoft technologies like .NET. It offers a user-friendly interface, making server management easier for less technical teams. However, licensing costs for Windows Server can be high, which could increase operational expenses for the client. While it's widely used for enterprise applications, Windows may not offer the same level of performance or customization as Linux for large-scale, high-traffic environments. | Mobile devices, such as iOS and Android, are not typically used for hosting web-based applications due to their limited processing power, storage, and networking capabilities. They are designed for client-side use rather than server-side deployment. While they can access web applications efficiently, their lack of scalability and high energy consumption makes them unsuitable as hosts for large-scale applications. |

| | | | | |
|---|---|---|---|---|
| **Client Side** | Developing for Mac requires expertise in macOS and potentially iOS, particularly if integration with Apple's ecosystem is desired. Costs may include licensing fees for macOS and tools like Xcode, which is essential for development. While macOS shares many web development standards, additional time may be needed to ensure compatibility and smooth user experiences across other platforms like Linux and Windows. | Linux development is cost-effective as the platform is open source, with no licensing fees for the OS or many development tools. Expertise in Unix-like systems, scripting languages, and web technologies like Apache or Nginx is required. Development time may vary depending on the need for cross-platform compatibility, but Linux's flexibility and extensive community support can reduce potential delays. | Windows development may involve higher costs due to licensing fees for the OS and development tools such as Visual Studio. Expertise in Windows-specific technologies, including .NET, IIS, and Microsoft’s web stack, is essential. Development time can be longer due to ensuring cross-platform compatibility, but Windows’ wide adoption and robust development environment streamlines many aspects of the process. | Developing for mobile devices entails costs related to platform-specific development tools and environments, such as Android Studio for Android and Xcode for iOS. Expertise in native programming languages (Java for Android and Swift for iOS) is necessary, along with knowledge of responsive web design to ensure compatibility across different screen sizes. Development time may increase due to the need for separate builds and testing for each platform, as well as compliance with app store guidelines. |

| | | | | |
|---|---|---|---|---|
| **Development Tools** | For Mac development, Swift and Objective-C are the primary programming languages used for native applications. Xcode is the main Integrated Development Environment (IDE) that provides tools for interface design, debugging, and testing. Additionally, web technologies such as HTML, CSS, and JavaScript are essential for creating responsive web applications that can run on macOS browsers, while frameworks like Electron can be utilized for cross-platform desktop apps. | Linux development primarily utilizes programming languages such as Python, C, C++, and Java, depending on the application requirements. Popular IDEs include Eclipse, Visual Studio Code, and JetBrains' IntelliJ IDEA, which support various languages and offer debugging and version control features. For web applications, technologies like HTML, CSS, JavaScript, and frameworks such as Node.js or Django are commonly used, alongside server software like Apache or Nginx for hosting. | Windows development often employs languages such as C#, C++, and VB.NET, especially within the Microsoft ecosystem. Visual Studio serves as the primary IDE, offering extensive features for debugging, testing, and deployment. For web applications, developers typically use HTML, CSS, and JavaScript, along with frameworks like ASP.NET and tools like IIS for hosting. Additionally, Microsoft Azure can be leveraged for cloud-based deployments, enhancing scalability and accessibility. | For mobile development, Java and Kotlin are primarily used for Android applications, while Swift and Objective-C are the languages of choice for iOS applications. Development environments include Android Studio for Android and Xcode for iOS, providing tools for app design, debugging, and testing. Additionally, cross-platform frameworks like React Native, Flutter, and Xamarin allow developers to write code that runs on both iOS and Android, utilizing shared web technologies such as HTML, CSS, and JavaScript for responsive designs. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: To effectively expand "Draw It or Lose It" across multiple computing environments, a cloud-based solution utilizing a Linux server platform is highly recommended. Linux is known for its scalability, allowing The Gaming Room to accommodate thousands of simultaneous players, which is essential for a web-based application. Its open-source nature significantly reduces licensing costs, making it a cost-effective choice for deployment. Furthermore, Linux has robust security features and extensive community support, ensuring that the application remains secure and receives timely updates. This flexibility allows seamless integration with Windows, Mac, and mobile platforms, providing a consistent gaming experience for all users.

2. **Operating Systems Architectures**: The chosen Linux server platform employs a modular architecture, consisting of the kernel, system libraries, and user-level applications. The Linux kernel is responsible for core functions such as managing hardware resources (CPU, memory, disk I/O), process scheduling, and multitasking, ensuring efficient operation even under high server loads. The architecture supports a microkernel structure that allows for minimal kernel functionality while delegating additional tasks to modules or user-space components, thus improving system stability and flexibility. System libraries provide essential application interfaces (APIs) for interacting with kernel services, including managing hardware and file systems. Additionally, the platform supports various server configurations, including web servers (using Apache or Nginx) and application servers, allowing for efficient deployment of the "Draw It or Lose It" game across different environments. This architecture ensures scalability, fault tolerance, and efficient resource allocation, enabling the platform to handle concurrent player requests and intensive data processing tasks smoothly.

3. **Storage Management**: For the recommended Linux server platform, a suitable storage management system is Logical Volume Management (LVM). LVM provides flexibility in managing disk storage by allowing the creation of logical volumes that can be resized dynamically as needed. This capability is essential for a growing application like "Draw It or Lose It," which may require adjustments to storage based on player data and game assets. LVM also enhances data protection through features like snapshots, enabling backups of data without downtime. By using LVM, The Gaming Room can efficiently manage storage resources while ensuring high availability and performance for their web-based game.

4. **Memory Management**: The recommended Linux server platform employs several memory management techniques to optimize performance for the "Draw It or Lose It" software. It utilizes virtual memory to extend the available memory space, allowing applications to operate efficiently even when physical RAM is limited. The Linux kernel implements paging and segmentation, breaking memory into manageable blocks that can be allocated or deallocated as needed, ensuring that the game can quickly access necessary resources. Additionally, memory caching techniques improve data retrieval speeds by storing frequently accessed data in faster memory areas. These techniques collectively enhance the responsiveness and stability of the application, providing players with a smooth gaming experience.

5. **Distributed Systems and Networks**: To enable "Draw It or Lose It" to communicate across various platforms, a distributed software architecture can be implemented using RESTful APIs for seamless interaction between the client-side applications and the server. REST APIs allow stateless communication, making it easy for various client devices—whether running on Linux, Mac, or mobile platforms—to send and receive game data over the network. To ensure robust network performance, a combination of load balancing, redundancy, and failover mechanisms should be implemented. Load balancing distributes incoming traffic across multiple servers to prevent any single server from being overwhelmed, improving both availability and response times. Redundancy ensures backup systems are in place to handle server failures without disrupting the gaming experience. Additionally, distributed database systems like Cassandra or MongoDB should be used to maintain data consistency and availability across geographically dispersed servers. These databases offer replication features, ensuring data remains synchronized even if a part of the system goes down. By employing these strategies, The Gaming Room can provide a stable and responsive gaming environment, even under varying network conditions or high player traffic.

6. **Security**: To protect user information across various platforms for "Draw It or Lose It," implementing robust security measures is essential. The recommended Linux server platform provides several security features, including user authentication and access controls, to ensure that only authorized personnel can access sensitive data. Data encryption is vital for safeguarding user information both at rest and in transit; utilizing protocols like TLS/SSL will secure communications between the server and client applications, preventing unauthorized access and data breaches. Additionally, regular security updates and patches can be applied to the Linux system to mitigate vulnerabilities. By adopting a multi-layered security approach, including firewalls, intrusion detection systems, and secure coding practices, The Gaming Room can protect user data effectively and enhance overall trust in their gaming platform.