The Gaming Room

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	05/25/2025	Katherine Rangel	Added changes to evaluation and recommended sections.

Executive Summary

The Gaming Room has requested a web-based version of their current game “Draw It or Lose It,” which is currently only available as an Android app. This game is similar in format to “Win, Lose or Draw,” where the players guess phrases based on drawings. To support the client’s expansion goals, the software solution must be capable of running across multiple platforms and meet specific software requirements. The proposed solution involves designing a modular, object-oriented system that utilizes the Singleton pattern which ensures that only one instance of the game is in memory at a time. It will also employ the Iterator pattern for managing collections of games, teams, and players. Each object must have a unique identifier, and the system must support multiple teams per game and multiple players per team. This design will be scalable and adaptable to future expansions.

Requirements

The client requires the following software functionality:

- A game can have one or more teams
- Each team can have multiple players
- Unique names for each team and games to avoid duplicates
- Only one instance of the game in memory at a time

These requirements must be implemented using object-oriented programming principles, along with the Singleton and Iterator design patterns.

Design Constraints

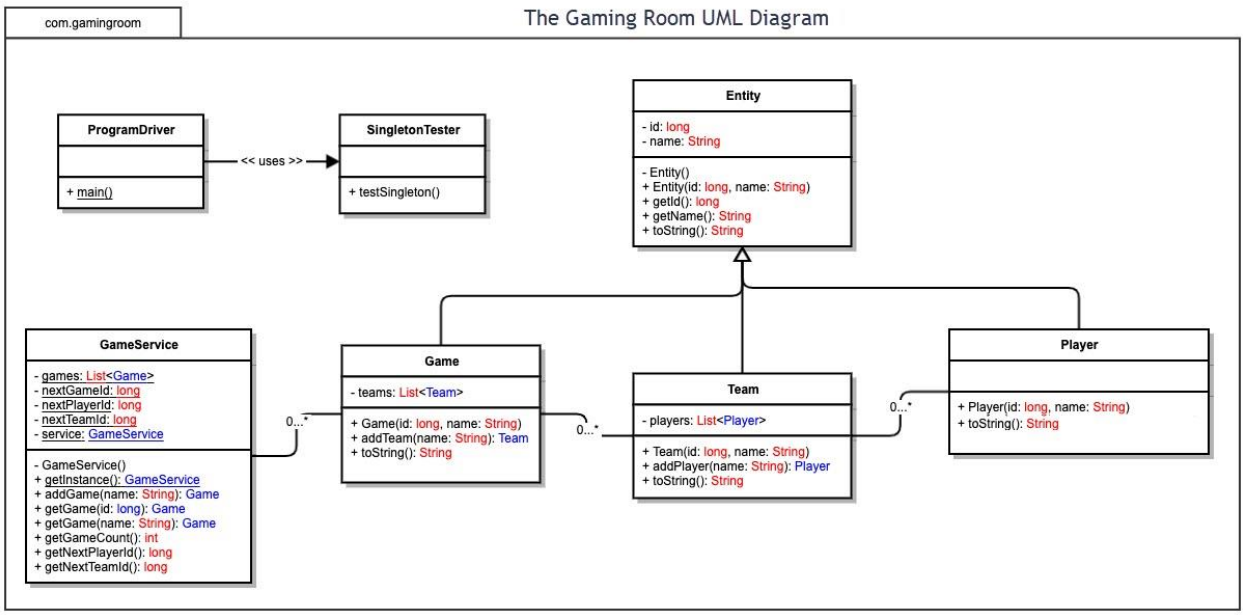
Th game must operate efficiently in a web-based distributed environment, which means it should be platform-agnostic and scalable. Memory management is crucial since there can only be one game instance existing at a time, which enforces using the Singleton pattern.

Naming conflicts must be avoided by checking for unique identifiers when adding games or teams. The system should also use the Iterator pattern to search through and manage collections, providing clean and maintainable code. These constraints will influence decisions such as which platforms to support, how data is stored in memory, and how objects interact within the system.

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



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 must 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	macOS is Unix-based and it supports many web technologies, but it's less commonly used for hosting production web applications. More suitable for development and testing.	Linux is the most popular server OS due to its stability, flexibility, and open-source nature. It's highly reliable for web hosting and production environments.	Windows Server is widely used in enterprise environments and supports many Microsoft technologies. But it does have higher licensing costs.	Mobile devices are not typically used as servers but can access and interact with web servers. It is limited in resources, power, and control for server hosting.

Client Side	This supports modern web browsers and Java applications. The development is smooth, but cross-platform testing is important.	Linux supports most web browsers and Java apps. It's developer-friendly, though not as common on client devices, which could lead to compatibility issues.	Windows is the most widely used desktop OS. High compatibility with games, browsers, and client-side applications.	Mobile devices are widely used by end users. Client-side performance varies by OS and device, requiring responsive design and testing.
Development Tools	Xcode, IntelliJ IDEA, Eclipse, and VS Code work well. macOS allows testing for IOS app directly. Terminal supports Git and Maven.	It has strong support for Jave tools like Eclipse, IntelliJ, and VS code. Ideal for server-side and open-source projects.	Broad support for Java and IDEs. Good integration with Windows - native tools.	Android Studio, Xcode. Limited file system access. Debugging and testing vary based on platforms.

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**: The recommended operating platform is Linux, specifically a distribution company such as Ubuntu Server or CentOS. Linux provides a stable, secure, and cost-effective environment for hosting web-based applications. It is widely supported in cloud platforms like AWS, Azure, and Google Cloud, which allow for future scalability.

2. **Operating Systems Architectures**: The chosen architecture should follow a multi-tiered client-server architecture, separating the user interface, application logic, and data management. This separation enhances maintainability, scalability, and allows for deployment across different environments. A RESTful API can serve as the communication layer between the client and the server tiers.

3. **Storage Management**: Using cloud-based storage solutions such as Amazon S3 or Google Cloud Storage for storing user data, game state, and media assets. These systems provide high availability, durability, and easy integration with scalable backend services. Additionally, consider using a relational database for structured data like users, games, teams, and scores.

4. **Memory Management**: The application will enforce a Singleton pattern for GameService to ensure only one instance of the game runs in memory at any given time. Proper object lifecycle management and garbage collection will help manage memory use efficiently. Data structures like List<T> will store game entities while minimizing overhead.

5. **Distributed Systems and Networks**: The system should use RESTful web services to communicate between the client and server. This allows platform-agnostic communication using

standard HTTP protocols. Hosting services can be load-balanced to handle high traffic, and content

6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>

PROJECT TWO TABLE:

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	macOS is Unix-based and supports many web technologies. However, it's less commonly used for hosting production-grade web applications. It can serve development and testing well but is not typically chosen for development due to licensing and scalability limitations.	Linux is the leading OS for web servers due to its open-source nature, strong stability, and flexibility. It is widely supported by cloud providers (AWS, Google Cloud), making it ideal for hosting scalable, secure, and cost-effective web applications.	Windows Server supports enterprise environments and Microsoft web technologies (.NET, IIS). While robust, it has higher licensing costs and is less favored for open-source deployments. It is best for applications relying on Microsoft's stack.	Mobile devices are not used for server hosting but must communicate with hosted backends. Server-side performance must consider mobile access constraints such as latency, screen size, and power consumption.
Client Side	macOS supports modern browsers and Java applications, with smooth development. However, software must be tested on other systems for compatibility. Not all users are on macOS, so testing across platforms is key.	Linux supports a range of browsers and applications. It's developer-friendly, but client compatibility testing is essential because not all end users run Linux, and browser behavior may vary slightly across distros.	Windows is the most used desktop OS, ensuring strong compatibility with most games, browsers, and client-side apps. It's ideal for broad deployment across desktop environments.	Monile platforms (iOS and Android) dominate client-side usage. Apps must be optimized for OS and device type. Each requires its own SDKs (Xcode for iOS, Android Studio), and cross-platform testing is essential.

Development Tools	macOS supports Xcode, IntelliJ, Eclipse, and VS Code. Developers benefit from Unix-like tools, including Terminal, Git, and Maven. It is ideal for building iOS apps and Java-based web systems.	Linux is favored by developers for its flexibility and support of open-source tools. It works well with Java-based IDEs like IntelliJ and Eclipse. It's ideal for full-stack development and automation with shell scription.	Windows offers broad IDE support, including Visual Studio, IntelliJ, and Eclipse. It is highly compatible with enterprise tools and has good integration with .NET and Microsoft-based platforms.	Android Studio is the main tool for Android apps, while Xcode is used for iOS development. Each has debugging tools, device simulators, and platform-specific SDKs. Cross-platform support is improving with frameworks like Flutter and React Native.

Recommendations

Based on the evaluation, Linux is the recommended platform for server-side hosting due to its scalability, security, and compatibility with cloud services. For client-side deployment, Windows is ideal for desktop use due to its market share and broad compatibility, while mobile deployment should support both iOS and Android platforms. For development tools, macOS is excellent for iOS development, Windows is strong for .NET environments, and Linux excels in open-source toolchains. A cross-platform approach using frameworks like Flutter or React Native will support maximum reach and flexibility.

PROJECT THREE:

Recommendation (Extended)

Operating Platforms

The recommended server-side operating platform is Linux due to its flexibility, security, cost-effectiveness, and strong community support. Linux distributions such as Ubuntu Server or CentOS are ideal for scaling the 'Draw It or Lose It!' application into multiple computing environments, including public and private clouds.

Operating System Architectures

A multi-tiered client-server architecture is ideal for this application. It separates the presentation layer (front-end web interface), application logic (game state, timer, session management), and the data layer (persistent storage of users, teams, and scores). This structure allows for modular updates, better scalability, and easier maintenance.

Storage Management

The recommended storage system is a cloud-based object storage solution such as AWS S3 or Google Cloud Storage. These platforms offer high availability, security, and integration with backend logic. In addition to object storage, a relational database such as PostgreSQL or MySQL should be used to manage structured data such as users, game states, and session history.

Memory Management

Linux employs efficient memory management strategies such as paging and virtual memory. The operating system dynamically allocates memory based on application needs, and features like memory caching and garbage collection (handled by the Java Virtual Machine or equivalent runtime) ensure that resources are reused effectively. This supports performance stability for real-time multiplayer gaming applications.

Distributed Systems and Networks

The application should be designed to run in a distributed system using RESTful APIs for communication between clients and servers. This allows the game to scale across multiple servers and handle increased traffic from concurrent users. Load balancing, redundancy, and failover mechanisms should be used to maintain availability. The network should support secure HTTP (HTTPS) for data transmission integrity and privacy.

Security

Security measures should include HTTPS for encrypted communication, authentication tokens (such as JWTs), role-based access control for players and administrators, and input validation to prevent injection attacks. Data at rest should be encrypted using standard encryption protocols (e.g., AES-256). Linux's built-in firewall and SELinux/AppArmor tools can enhance system-level protection.