THE GAMING ROOM

Draw It or Lose It – Web Application

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/16/2025	Sydney Brown	-Initial draft of the full software design document completed.
1.1	11/30/2025	Sydney Brown	-Finalized the Evaluation and Recommendations sections and completed the full Software Design Document for Project Two
1.2	12/13/2025	Sydney Brown	- Submitted existing Software Design Document for Project Three. Recommendations section reviewed and confirmed to meet Project Three requirements.

Executive Summary

This project takes the existing Android game, Draw It or Lose It, and prepares it for a web-based version that different users can access at the same time. The biggest things the client needs are: one single instance of the game running, unique names for games, teams, and players, and a structure that can grow as the project gets bigger. The design uses a Singleton GameService to control all game data, plus simple object-oriented ideas like inheritance and lists to organize everything. Overall, the design gives the client a solid starting point that's clean, organized, and easy to build on.

Requirements

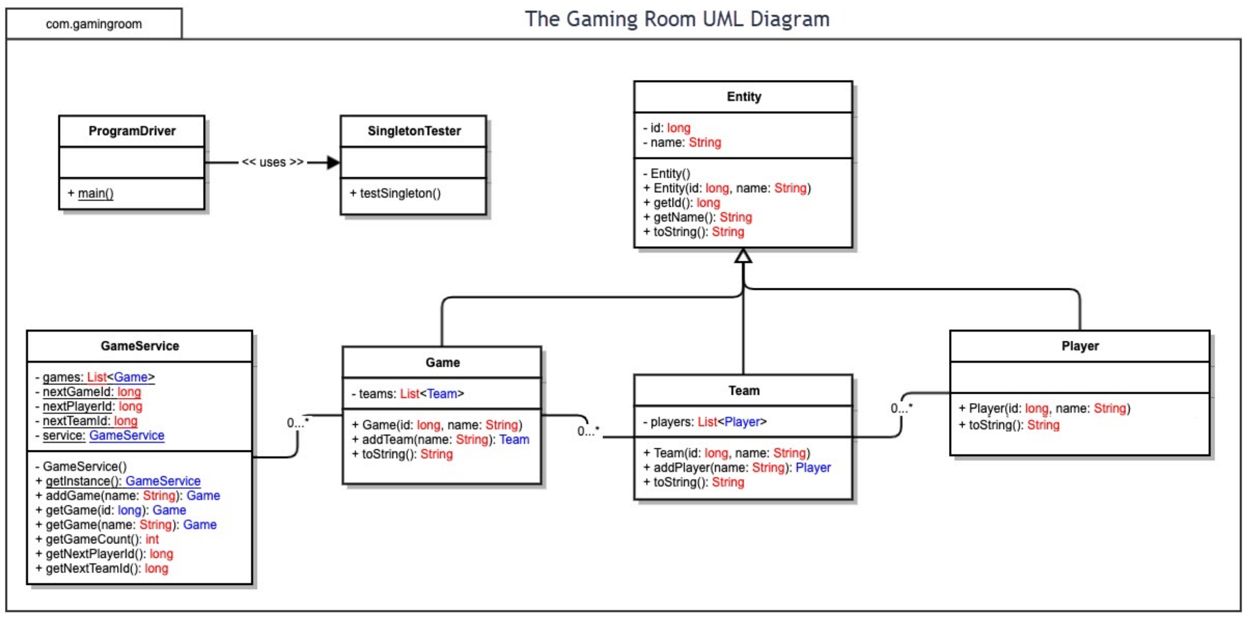
The client needs several things for the web-based version of the game. First, the system has to support multiple teams, and each team needs to allow multiple players. All names—games, teams, and players—must be unique to avoid duplicates. Only one instance of the game can exist in memory at a time, which means there must be a central service controlling everything. Since the new version will run on the web, the application also needs to support multiple users connecting from different platforms. These are the key business and technical requirements the design must meet

Design Constraints

The biggest constraint is making sure there's only one game instance running. That's why the GameService class uses the Singleton pattern. Another constraint is that everything has to work in a web-based environment, meaning the server has to keep all the "real" game data, while users might be connecting from anywhere. The system also has to enforce unique names for games, teams, and players. And finally, the design needs to work across different operating systems, since the final deployment isn't tied to one type of machine.

Domain Model

The UML diagram shows how the main pieces of the game fit together. All classes inherit from the Entity class, which gives them a shared structure for IDs and names. A Game holds Teams, and Teams hold Players, so it's built using composition. GameService manages all of this and uses the Singleton pattern so there's only one place the game data is controlled. The lists and iterators help search for items and prevent duplicates. Overall, the diagram uses basic object-oriented principles to keep the design clean, organized, and easy to expand.>



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 can run a web-based application, but it’s not usually used for hosting. It’s stable and easy to work with for development, but it’s more expensive and not ideal for large-scale deployment. It works, but it’s not the best long-term server option.	Linux is the strongest server choice because it’s reliable, secure, lightweight, and free. Most web servers run on Linux for a reason. It handles Java applications well and scales easily for multiple users.	Windows can host web applications, and it works fine with Java, but it comes with licensing costs and heavier system requirements. It’s solid, just not as cost-effective or flexible as Linux.	Mobile devices aren’t used as servers. They can connect to the game, but they can’t host it. This platform only works on the client side
Client Side	Mac is easy to support as a client because users only need a browser. Development and testing are simple, but the hardware is more expensive. Overall, it works well for client access.	Linux clients work fine through any modern browser. It’s less common for everyday users, but cheap and flexible for development and testing	Windows is the most common client platform, so supporting it is important. Browsers work smoothly, and testing is straightforward. Hardware is affordable and widely used.	Mobile clients work as long as the web app is mobile-friendly. The main consideration is adjusting the UI for small screens and touch input. No major extra cost, just design work.
Development Tools	Developers can use Java, Eclipse or IntelliJ, and standard web tools on macOS. Everything needed to build and test the application works well here	Linux supports Java, Eclipse, VS Code, and other dev tools with no issues. It’s lightweight and great for both development and deployment, which makes it a strong option.	Windows also works well with Java, Eclipse, IntelliJ, and other build tools. It’s widely used and has good support for development environments.	Mobile devices aren’t used to develop or deploy the app. They’re only for accessing the game, so no development tools apply here.

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**: I recommend using Linux as the operating platform. It's reliable, secure, and widely used for hosting web applications, especially ones built with Java. It's lightweight, free, and easy to scale, which makes it a strong fit for the long-term growth of The Gaming Room's web-based version of Draw It or Lose It.

2. **Operating Systems Architectures**: Linux supports a simple and clean architecture for web applications. It handles Java runtimes well, and most server tools and web technologies are already optimized for it. The system is flexible, works on many different hardware setups, and is stable enough to support multiple users connecting at once without unnecessary overhead.

3. **Storage Management**: For storage, a standard relational database like MySQL or PostgreSQL would work well with Linux. These databases are stable, open-source, and easy to integrate with a Java application. They allow the server to store game data, team names, and player details safely and efficiently as the project expands beyond in-memory storage.

4. **Memory Management**: Linux handles memory efficiently and works smoothly with Java's built-in garbage collection. This makes it easy for the system to manage objects like games, teams, and players without manually freeing memory. Only having one GameService instance also keeps memory usage predictable, which helps keep the application stable.

5. **Distributed Systems and Networks**: Since users will be connecting from different devices, the application will rely on web requests to communicate with the server. Linux handles this well because it's built for network-heavy workloads. As long as the server uses a stable network setup and handles outages or slow connections gracefully, the distributed system will work smoothly. Each device acts as a client, and all real data stays on the Linux-based server.

6. **Security**: Security is important since multiple users will be interacting with the game. Linux already provides strong built-in security features. The application should run over HTTPS, validate all user input, and store as little personal information as possible. Access to the server should be limited, updates should stay current, and the database should be protected behind proper authentication. Overall, Linux gives a solid security foundation to build on