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	07/20/25	Preston Mesecher	Document created for initial review
2.0	08/03/25	Preston Mesecher	Addition of Development Requirements
3.0	08/17/25	Preston Mesecher	Addition of 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 has an Android game called *Draw It or Lose It* and wants to grow by making it available on the web. Their goal is to reach more users by expanding to a web-based version so people can play on different devices. Our solution will support multiple teams, multiple players on those teams, prevent duplicate names, and make sure only one game runs at a time using unique IDs. This setup keeps things organized and sets us up for the next steps in development.

Requirements

The Gaming Room wants a web version of Draw It or Lose It that works on multiple platforms, expanding from their app currently only for Android. The game needs to support several teams, each with multiple players, use unique names, and allow only one game to run at a time. The design must be simple and efficient to meet their goals.

Design Constraints

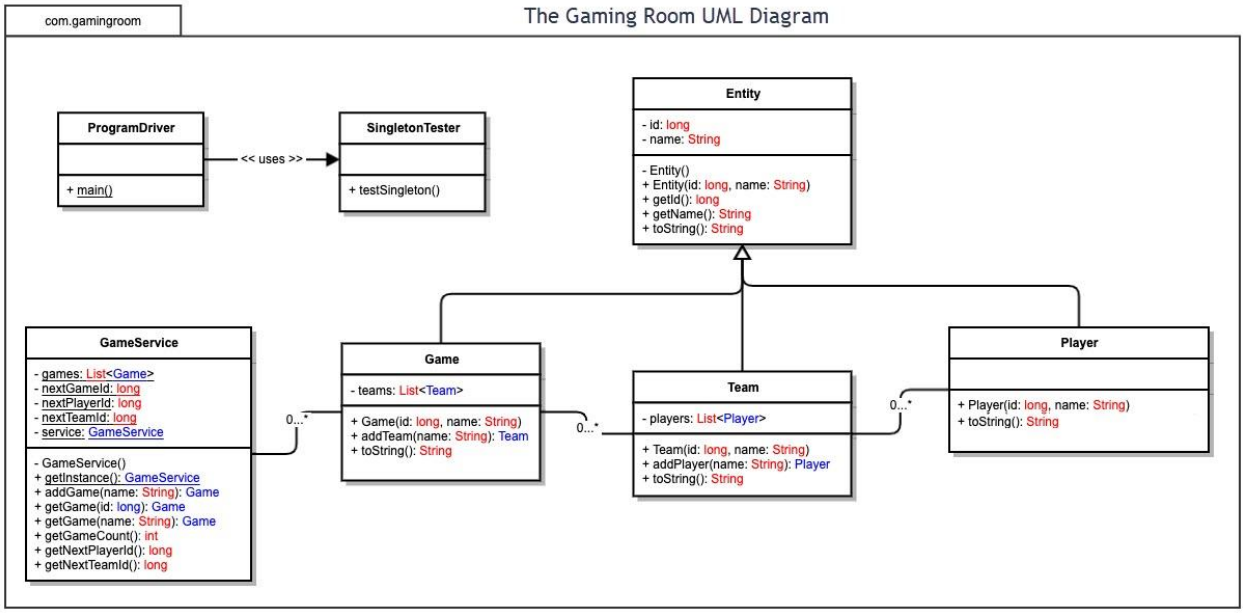
The game must work across different devices, operating systems, and browsers, support multiple users at once, and run smoothly. These requirements limit how we can build and test the app to keep it fast, simple, and reliable for every user and use-case.

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 diagram shows that GameService controls the game and uses a Singleton pattern to make sure only one game runs at a time. Game, Team, and Player inherit from Entity, which gives them shared fields like id and name. A Game has multiple Teams, with each Team having multiple Players. This setup meets the software requirements and keeps the structure simple and efficient. It also demonstrates encapsulation, as each class manages its own data and behavior. It demonstrates abstraction by hiding more complex details like ID management. Polymorphism is supported through Game, Team, and Player all extending from Entity, allowing them to be handled the same way when using shared features, even though they represent different object types.



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	Can be used to host websites, but not ideal for large scale apps. It's more expensive and harder to scale than other options.	Supports server based hosting and is widely used for websites and web apps. There are no licensing costs, which makes it the most cost effective option.	Supports server deployment through IIS for Windows Server, which adds to moderate costs for licensing.	Not suitable for server deployment. Phones and tablets lack the hardware and operating system needed to host web applications.
Client Side	Requires testing to ensure compatibility with macOS web browsers. Developers need access to Apple hardware. May slightly increase time and cost.	Works well with major web browsers. No special development tools are needed. Minimal extra effort required, keeping time and cost low.	Needs thorough browser testing due to wide usage. Development is straightforward, but additional testing time is expected.	Requires responsive design and testing across many screen sizes and mobile operating systems. Native apps increase complexity unless using cross-platform tools.
Development Tools	Uses Swift for Apple apps. Also supports Python, C++, and Java. These languages are learned in college or coding bootcamps. May need a separate team with Apple computers.	Uses Python, Go, C++, PHP, and Java. These are free and work on most systems. Easy to share tools across teams and keep costs low.	Commonly uses C# and SQL, as well as Java, Python, and JavaScript. Some tools may cost money and might need a Windows focused team.	Android uses Java and iOS uses Swift. These are taught in bootcamps and college. May need two teams unless using shared tools like JavaScript.

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**: Linux is the best choice for running the game. It's stable, free to use, and works well for web apps. It also makes it easy to scale up later if more players join, and it runs smoothly on cloud services like AWS or Azure.

2. **Operating Systems Architectures**: Linux is built to handle lots of users and processes at once. It's modular, which means parts can be updated or changed without breaking the whole system. This setup keeps the game reliable and flexible.

3. **Storage Management**: The best option is cloud storage with Linux's file system. Cloud services such as Azure can handle backups, growth, and reliability. A database like SQL will store player info, game sessions, and scores.

4. **Memory Management**: Linux manages memory really well using virtual memory. It gives active processes priority and can swap out idle ones. It also uses caching to speed things up and frees memory that's no longer needed, keeping gameplay smooth.

5. **Distributed Systems and Networks**: The game will use a client-server setup. The Linux server handles the game, while players connect from browsers or mobile devices. Communication happens through REST APIs over HTTPS. Traffic is shared across servers, and backup servers keep the game running if one fails.

6. **Security**: Linux has good built-in tools for managing access. The game will use HTTPS to encrypt data, and the database will encrypt info at rest. Input checks will help block hacks and keep bad data out. Regular updates keep things safe, and multi-factor login can be added for extra protection.