<Draw It or Lose It>

CS 230 Project Software Design

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	<01/23/2025>	<Prithvi Ghale>	Initial version of the software design document with the executive summary, design constraints, domain model, platform evaluation, and recommendations for The Gaming Room project.

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” wants to expand their game Draw it or lose it which is an Android app and want to turn it into a web-based game that works for multiple platforms. This document talks about how to design the game to meet their needs. The solution includes using software patterns like Singleton to make sure only one instance of the game exists and an iterator to make sure that it has a unique name for games, teams, and platters. These steps will help make the game scalable, secure, and able to work across different devices. Linux was chosen as the operating platform due to its reliability, scalability, and cost-effectiveness, making it ideal for hosting the game across multiple platforms.

Requirements

Some requirements for the game to support are:

- The game must support multiple teams, and each team can have several players in them. This allows players to have group participation.
- Each team and game names must be different from each other to prevent users from confusion and allow easy identification.
- At a time only one version of the game should be running in memory to make sure it has a smooth performance, and this allows us to avoid conflicts.
- Each game, team, and player must have a unique ID to properly track and manage data.
- The game must work on web-based platforms, including Windows, Linux, and mobile devices, this helps to reach a wider audience.
- The game must handle 200 high-definition images, each 8 MB in size, requiring efficient memory and storage management.

Design Constraints

The design constraints:

- Distributed System: The game must allow it to run on different platforms at the same time, this means the system must handle communication between devices.

- Real-Time Updates: The game should allow teams to play and interact in real time without delays.

- Unique Identifiers: Every game, team, and player must have a unique ID and name, which means extra steps to check for duplicates.

- Cross-Platform Compatibility: The design must work on both desktops and mobile devices, so it needs flexible programming and testing.

- Memory Optimization: The game must use caching and memory optimization techniques to handle 200 high-definition images efficiently.

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.

<u>Domain Model</u>

The UML diagram shows how the different parts of the game application work together.

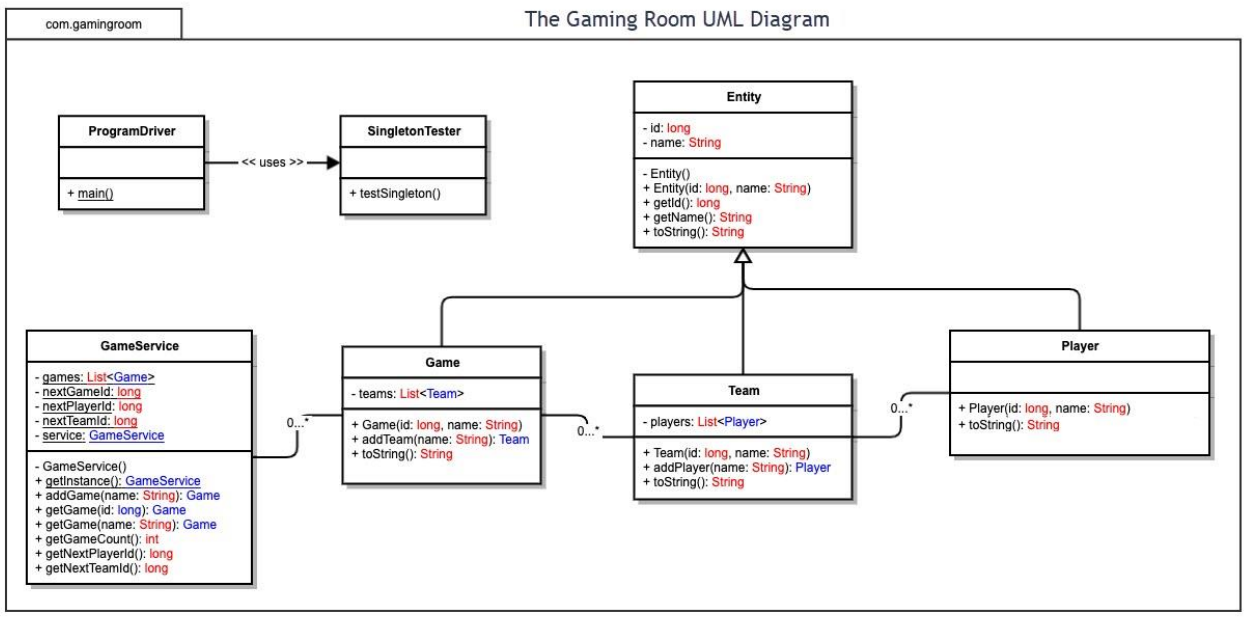
1. The classes and relationships: Here the entity class is the main base class with common attributes like id and name. Other classes like Game, Team, and Player inherit from this class, so they share these attributes. The Game Service class is responsible for creating and managing games. It makes sure there is only one instance of the service running by using the Singleton pattern. The Game class has a list of Team objects, which means one game can have multiple teams. The Team class has a list of Player objects, meaning each team can have multiple players. Program Driver and Singleton Tester are used to run and test the application.

2. Object-Oriented Principles:

- Encapsulation: The id and name attributes are private, and public getter methods are used to access them. This ensures data is controlled and secure.
- Inheritance: The Game, Team, and Player classes inherit from Entity, allowing them to reuse the same attributes and methods. This makes the code cleaner and reduces duplication.
- Polymorphism: Each class overrides the toString() method to display custom information about objects, depending on whether it's a game, team, or player.
- Composition: The Game class includes a list of Team objects, and Team includes a list of Player objects. This structure allows flexibility in organizing teams and players within games.

3. How It Meets Requirements: The Entity class ensures that all objects (games, teams, and players) have unique IDs, which are managed by GameService. The Singleton pattern in GameService makes sure that there is only one instance of the service in memory, preventing any

conflicts. The relationships between Game, Team, and Player allow the system to handle multiple teams and players within a game efficiently. Methods like addGame() and addTeam() include checks for unique names, ensuring no duplicates for games or teams.



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 host web apps but is not commonly used for servers because it’s expensive and not as flexible as Linux. It’s better for testing or small setups.	Linux is the best choice for hosting web apps because It’s reliable, free, and works well for big systems. Many companies use Linux for servers.	Windows is good for hosting but costs more than Linux. It works well with business tools but needs more resources.	Mobile devices aren’t used for hosting servers. They’re mainly used to accessing the app as clients. They don’t have the power to handle server tasks.

Client Side	Building apps for Mac is simple if you have Apple devices, but it's expensive because Macs are costly, and you need tools like Xcode.	Linux has fewer users, so supporting it isn't a big focus. It's lightweight and good for certain users but not common for regular clients.	Windows is widely used, so it's important to support. Many people use Windows, but it takes more time and resources to make it work for all versions.	Most users access apps through mobile phones, so it's very important to support Android and iOS. You'll need tools like Android Studio for Android and Xcode for iOS.
Development Tools	Mac uses Xcode for macOS and iOS apps, but you can also use IntelliJ IDEA or Eclipse for general development. It's good for building apps but limited to Apple systems.	Linux supports many free tools like Eclipse and IntelliJ IDEA. It works well with programming languages like Java and Python, making it great for development.	Windows has a lot of tools like Visual Studio, IntelliJ IDEA, and Eclipse. It supports most programming languages and is a flexible choice for developers.	For mobile development, you use Android Studio for Android apps and Xcode for iOS. Tools like Flutter or React Native let you build apps for both platforms at the same time.

<u>Recommendations</u>

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: An appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments is Linux. Linux is free, reliable, and widely used for hosting web applications. In Linux it offers scalability, making it ideal for expanding Draw it or Lose it to other platforms like Windows, Mac, and mobile devices. It also supports a wide range of programming languages and tools for web-based development.

2. Operating Systems Architectures: Linux uses a modular and flexible architecture that allows the system to handle multiple tasks efficiently. It supports multi-threading, which is essential for running a web-based game with real-time interactions.

3. Storage Management: For storage, MySQL or PostgreSQL are recommended because they are compatible with Linux and are powerful for managing relational data like games, teams, and players. These databases can handle large volumes of data and support quick retrieval, which is critical for real-time gameplay.

4. Memory Management: Linux uses advanced memory management techniques such as caching and virtual memory to optimize performance. It efficiently allocates memory to processes, this helps to make sure that the game runs smoothly even with many players interacting at once. The

Singleton pattern in the application also helps minimize memory usage by making sure only one instance of GameService exists.

5. Distributed Systems and Networks: The game can communicate between platforms using REST APIs, which allows data to be sent and received across devices. A distributed system like this relies on the internet for connectivity, so making sure there is a low-latency server and backup systems is key to maintaining reliability. Load balancers can be used to distribute traffic evenly, and redundancy ensures the system stays functional during outages.

6. Security: To protect user information, use SSL/TLS encryption for secure data transmission between platforms. The system should also implement authentication and authorization, such as password protection and token-based authentication (like OAuth). For database security, enable encrypted storage and restrict access to sensitive information. Regular updates and monitoring should be done to protect against vulnerabilities and attacks.