Draw It or Lose It – Web Based Game Application

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	01/23/26	Cindi Rodriguez	Outlining the proposed solution for the Draw It or Lose It web-based game application, including software requirements, design constraints, and model overview.
2.0	02/17/26	Cindi Rodriguez	Evaluate Operating Platforms
3.0	02/17/26	Cindi Rodriguez	Architecture 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 requested assistance in expanding their existing Android game, Draw It or Lose It, into a web-based application that can support multiple platforms. The goal of this project is to design a scalable and efficient software solution that allows multiple games, teams, and players to interact while ensuring that system resources are managed correctly.
To meet these needs, this project proses a software design that uses object-oriented principles and established design patterns to create a structured and maintainable application. A singleton pattern is used to ensure that only one instance of the game service exists in memory at any given time, preventing data conflicts and ensuring consistency across the application. Iterator patterns are implemented to enforce unique game and team names to efficiently search existing data.
The proposed design provides a strong foundation for future expansion into additional platforms while maintaining performance, security, and ease of maintenance. This document outlines the design constraints, domain models, and recommended technologies needed to successfully develop the web-based version of Draw It or Lose It.

Requirements

The Gaming Room's web-based game needs to allow multiple games to run, each with one or more teams and multiple players per team. Game, team, and player names must be unique, so users don't get confused when joining or creating games. Only one instance of the GameService should exist in memory at a time to keep everything consistent. The system should be able to run across multiple platforms and handle multiple users at once, meaning it needs to be stable, scalable, and easy to maintain.

Design Constraints

1. Singleton GameService – only one instance of GameService can exist in memory at a time. This keeps all games, teams, and players consistent and prevents duplicate data from appearing across the system.
2. Unique Names – Game, team, and player names must be unique. The system uses the iterator pattern to check for existing names before adding a new game, team, or player which keeps things organized and prevents confusion.
3. Web-Based Distributed Environment- the game has to work across multiple platforms and handle multiple users at the same time. The design must account for network communication, syncing game data, and scaling for future growth, so it stays stable and reliable.

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 communication and storage aspects is also necessary to understand the overall architecture and should be provided.

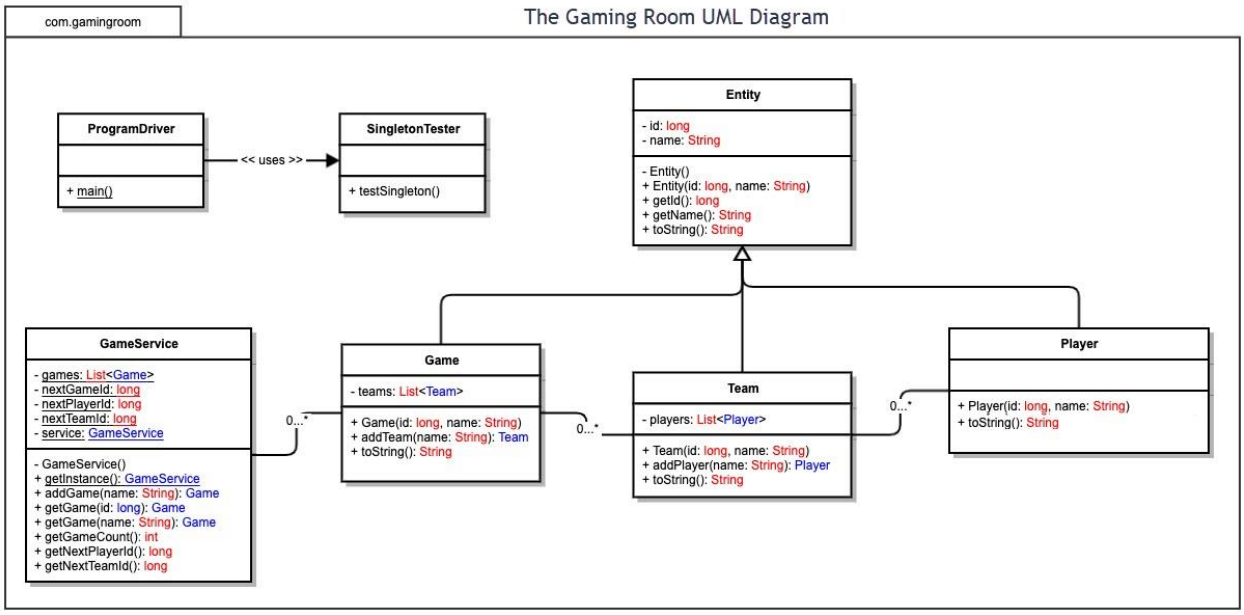
Domain Model

The UML diagram shows the core classes for the Draw It or Lose It game. At the top of the entity class, which holds shared attributes like id and name. This makes it easy for other classes to inherit common functionality without repeating code.

The Game class inherits from Entity and contains multiple Teams, while each Team can have multiple Players, which also inherit from Entity. This hierarchy shows a clear parent-child relationship and keeps the design organized.

The GameService class manages all the games, teams, and players. It uses the Singleton pattern to make sure only one instance exists in memory, and the Iterator pattern to efficiently check for unique names when adding games, teams, or players.

Overall, the UML demonstrates inheritance (Entity -> Game/Team/Player), encapsulation (private attributes with access methods), and abstraction (shared functionality in Entity is hidden from child classes), which helps the application stay scalable, maintainable, and efficient while meeting all the client's requirements.



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 stable and secure, but they are not commonly used as production servers. They are expensive and don't scale as well as other options, making them better for development than hosting a web-based game.	Linux is the most popular choice for hosting web-based applications because it is reliable, secure, and cost effective. It scales well and is ideal for running a distributed multiplayer game like Draw It or Lose It.	Windows servers can host web applications and work well with Microsoft tools, but they often come up with higher licensing costs. While reliable, they are usually less cost efficient than Linux for server hosting.	Mobile devices are not designed to host servers due to limited performance and reliability. They are meant for users to play the games, not run the backend infrastructure.
Client Side	Mac computers offer strong performance and security for users and developers. However, higher hardware costs can limit accessibility for some players.	Linux is affordable and flexible, but it usually requires more technical knowledge from users. This can make it less user-friendly for a general gaming audience.	Windows is the most widely used platform, making it easy to reach a large number of users. Development tools are widely available and familiar, which help reduce development time.	Mobile devices provide the most accessibility and allow users to play from anywhere. Supporting mobile platforms is essential for expanding the games user base.
Development Tools	Mac supports popular development tools like Eclipse, IntelliJ, and Xcode. It is especially useful for cross-platform and mobile development.	Linux supports a wide range of open-source development tools and server frameworks. It is commonly used for backend development and deployment.	Windows supports major IDEs like Eclipse and Visual Studio. It is widely used in professional environments and easy to develop on.	Mobile development requires specialized tools like Android Studio and Xcode. These tools are necessary to build and deploy apps for mobile platforms.

Recommendations

1. **Operating Platform**: I recommend a web-based platform that works across Linux, Mac, and Windows for desktop users, and Android/iOS for mobile users. This ensures that Draw It or Lose It can reach the largest audience while remaining consistent and easy to maintain.

   The most appropriate server platform for expanding Draw It or Lose It across multiple computing environments is Linux deployed within a cloud-based infrastructure. Linux provides strong reliability, security, scalability, and cost efficiency, making it the industry standard for distributed web applications. Clous hosting enables the system to support thousands of concurrent users while maintaining high availability and performance, which is essential for a multiplayer game environment.

2. **Operating Systems Architectures**: The chosen platforms should use standard client-server architecture, with the game service running on a central server and clients connecting via web browsers or mobile apps. This allows the system to handle multiple users and devices efficiently.

   Linux operates using monolithic kernel architecture with modular components that efficiently manage hardware resources, processes, memory, and networking. It also supports virtualization and containerization technologies such as virtual machines and Docker containers, allowing multiple game service instances to run simultaneously. These architectural capabilities enable horizontal scaling, fault tolerance, and efficient resource utilization required for distributed gaming systems.

3. **Storage Management**: Use centralized storage for all game, team, and player data. A relational database like MySQL or PostgreSQL works well, ensuring data is consistent, secure and easily queried by the GameService.

   A centralized relational database system, such as MySQL or PostgreSQL, should manage persistent game data including players, teams, scores, and sessions. High-definition image assets should be stored using cloud object storage or a content delivery network (CDN) to improve delivery speed and reduce server workload. Redundant backups, replication, and secure access controls ensure long term reliability and data protection.

4. **Memory Management**: The Singleton pattern for GameService keeps memory usage by ensuring only one instance manages all games. Each game, team, and player object is created only when needed, and iterators help avoid unnecessary searches or duplicates.
   Linux uses virtual memory, paging, and process isolation to efficiently manage RAM and prevent system instability. For Draw It or Lose It, techniques such as lazy loading of images, in memory chaching of frequently used assets, and object reuse help maintain fast rendering speeds without excessive memory consumption. These strategies ensure consistent performance across desktop and mobile clients even during simultaneous gameplay sessions.

5. **Distributed Systems and Networks**: The game should use a distributed design where the central server communicates with multiple clients. Using REST APIs or WebSocket ensures real time updates, keeps game state synchronized and allows the application to scale as more users join.

   The application should follow a distributed client server architecture in which centralized Linux servers communicate with desktop and mobile clients using REST APIs or WebSocket connections over HTTPS. Load balancers distribute incoming traffic across multiple server instances to maintain responsiveness and availability. This architecture supports

scalability, synchronization of real-time gameplay, and resilience against connectivity interruptions.

6. **Security**: Protect user data by implementing standard security measures like HTTPS for network communication, server-side validation, and secure storage for any sensitive information. This prevents data breaches and ensures a safe, trustworthy experience for players across all platforms.

Protecting user information requires end-to-end security controls across all platforms. HTTPS encryption secures network communication, while authentication and authorization mechanisms restrict access to game services and player data. Server-side validation prevents malicious input, and secure database practices protect stored information. Linux security features such as permission controls, firewalls, and regular patching further strengthen system protection and ensure a safe gaming environment.