

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/24/2024 | Samuel Alexander Black | Initial draft for submission. |
| 1.1 | 12/08/2024 | Samuel Alexander Black | Updated evaluation and requirements, reviewed previous content |
| 1.2 | 12/21/2024 | Sameul Alexander Black | Revising Recommendation section to include more  information specific to the client |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room seeks to expand its mobile game, "Draw It or Lose It," to a web-based platform accessible on multiple devices, including desktops, laptops, tablets, and smartphones. This game, inspired by the TV game show Win, Lose or Draw, challenges teams to guess clues rendered from a stock image library. Expanding to a web-based environment will enhance accessibility, broaden the user base, and improve the overall gaming experience.

This document outlines a software design solution to address the client’s requirements and constraints, focusing on efficiency, scalability, and security to ensure the game operates seamlessly across various platforms.

## Requirements

1. *Web Accessibility: The game must be accessible through a web-based platform compatible with multiple devices and operating systems.*
2. *Team and Player Structure: Each game will support one or more teams, with multiple players assigned to each team.*
3. *Unique Identifiers: Game, team, and player names must be unique to prevent conflicts during game setup.*
4. *Singleton Design: The application must enforce a single instance of the GameService class in memory at any given time.*
5. *Game Rounds: Each game consists of four rounds lasting one minute each, with drawings fully revealed at the 30-second mark.*
6. *Guessing Mechanics: If a team fails to guess a puzzle, other teams can submit one guess each within a 15-second time limit.*

## [Design Constraints](#_2et92p0)

1. Web-Based Distributed Environment: The game must function in a distributed environment, requiring efficient network communication, compatibility with web browsers, and strong security protocols. These constraints influence the selection of technologies for development.
2. Unique Naming Enforcement: The system must ensure unique game, team, and player names. This constraint necessitates the use of the iterator design pattern to search for and validate names within the game service.
3. Singleton Pattern: To maintain only one instance of the GameService class in memory, the singleton pattern must be implemented. This design constraint ensures centralized management of game data and prevents duplication.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

* **Entity Class: Serves as the base class for Game, Team, and Player, encapsulating shared attributes like id and name.**
* **GameService Class: Implements the singleton pattern to ensure a single instance manages all active games. It maintains composition relationships with Game, Team, and Player classes.**
* **Game Class: Contains a collection of Team objects and enforces the unique naming constraint for games.**
* **Team Class: Represents groups of players within a game, maintaining a list of Player objects and enforcing unique team names.**
* **Player Class: Represents individual participants, encapsulating attributes like id and name.**

**Object-Oriented Programming Principles**

* **Inheritance: The Entity class is a superclass for Game, Team, and Player, promoting code reuse and consistency.**
* **Encapsulation: Each class restricts direct access to its attributes through getter and setter methods, ensuring data integrity.**
* **Abstraction: High-level operations like managing games and teams abstract unnecessary implementation details from the user.**
* **Polymorphism: Methods in derived classes override or extend those in the base class to support specialized behaviors.**

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| Server Side | macOS offers excellent integration with a graphical user interface and robust server configurations. It is user-friendly and supports multitasking, though it can be limited by hardware upgradeability. Server costs are moderate, and it excels in creative or design-centric environments. | Linux is highly cost-effective, open-source, and customizable. Its flexibility supports a wide variety of server needs, but it requires more technical expertise. Known for reliability and security, Linux is ideal for enterprises, though pre-built hardware options are limited. | Windows provides seamless Active Directory integration and an intuitive interface for server management. While it supports diverse applications, licensing costs can be high, and its security is not as robust as Linux. Windows servers are common in corporate environments. | Mobile devices require compact, energy-efficient applications with lighter server-side resources. They are generally less secure than desktops, and mobile server compatibility can vary significantly depending on the device and operating system. |
| Client Side | Development on macOS requires skilled expertise, as its ecosystem has specific requirements for optimization. Costs can be high, but macOS provides a smooth and stable user experience. It is preferred for creative industries due to its design tools. | Linux provides flexibility and affordability but demands significant expertise. Development on Linux is highly customizable, but compatibility and support for certain tools can be challenging, making it better suited for technical users. | Windows development is beginner-friendly, with broad compatibility across platforms. Its tiered systems allow for varied costs, but development can be expensive for enterprise applications. It provides an intuitive development environment but requires careful testing across devices. | Mobile app development is highly accessible, with a wide range of tools available for Android and iOS. Applications must be optimized for different screen sizes and performance constraints. Developing for multiple mobile platforms adds complexity but increases user reach. |
| Development Tools | macOS relies on development tools like Xcode, which is specifically tailored for iOS and macOS applications. These tools offer robust features but come with licensing fees. Familiarity with Swift and Objective-C is necessary for macOS application development. | Linux supports a variety of development environments like Eclipse, NetBeans, and Vim. Its open-source nature reduces licensing costs, but teams need expertise in languages such as C, C++, and Python for effective development. | Windows offers a wide range of tools like Visual Studio, which supports multiple languages such as C#, .NET, and Java. While tools like Visual Studio are powerful, they often come with significant licensing costs for enterprise editions. | Mobile platforms use development tools like Android Studio for Android and Xcode for iOS. Cross-platform tools such as Flutter or React Native allow developers to create apps for both platforms, though they require familiarity with Java, Swift, and JavaScript. |

## Recommendations

1. **Operating Platform:**  
   The recommended operating platform for "Draw It or Lose It" is Linux, as it is cost-effective, highly secure, and versatile for server hosting. Linux's open-source nature allows for extensive customization, making it ideal for web applications. Its stability and low hardware requirements provide an advantage when scaling the server infrastructure.
2. **Operating Systems Architectures:**  
   The Linux operating system architecture is built on a modular design, divided into the kernel, system libraries, and user-space applications. The kernel manages core functionalities such as memory, process management, and networking. Linux employs a monolithic kernel, which allows for efficient interaction between hardware and software, critical for hosting high-traffic web-based applications.
3. **Storage Management:**  
   A hybrid storage management solution is recommended:
   * **Relational Database (MySQL):** To manage structured data like player profiles, game states, and team information.
   * **Object Storage (Amazon S3):** To store and retrieve the 200 high-definition image files efficiently. Object storage is scalable, ensuring the game assets are accessible globally without performance degradation.
4. **Memory Management:**  
   Linux utilizes advanced memory management techniques such as paging and virtual memory, enabling efficient use of RAM for rendering images and game processes. It supports on-demand memory allocation, allowing resources to be assigned only when needed, thus optimizing performance for simultaneous game sessions.
5. **Distributed Systems and Networks:**  
   The distributed architecture will rely on a centralized server to synchronize game data across devices. Communication between clients and servers will use RESTful APIs for secure and efficient data exchange. WebSockets are recommended for real-time updates, such as team guesses and leaderboard updates. To enhance fault tolerance, load balancers and redundancy mechanisms will ensure high availability during outages or peak traffic.
6. **Security:**  
   To protect user information, the following measures are recommended:
   * **Encryption:** Implement HTTPS for secure data transmission and AES for encrypted storage of sensitive information like user credentials.
   * **Authentication:** Use OAuth 2.0 for secure login and session management.
   * **Access Control:** Role-based access ensures that users only access resources relevant to their permissions.
   * **Intrusion Detection and Firewalls:** Employ tools like Fail2Ban and UFW (Uncomplicated Firewall) to monitor and block malicious activities.