

Win, Lose or Draw

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 09/19/2024 | Katie Moore | Initial draft of the software design document |
| 1.1 | 10/6/2024 | Katie Moore | Development requirements and recommendation edits. |
| 1.2 | 10/15/2024 | Katie Moore | Techniques specific to system architecture under the requirements section |

**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](#_sbfa50wo7nsh)

The Gaming Room aims to expand its existing mobile game, Draw It or Lose It, into a web-based application accessible across multiple platforms. To avoid conflicts, the primary challenge involves developing a scalable and robust system that supports numerous teams and players, with unique identifiers and names for each entity. This design document outlines the solution to address these requirements, including implementing software design patterns such as singleton and iterator patterns to ensure a single instance of the game and efficient management of game and team names. The document also addresses design constraints and provides system architecture and development tools recommendations.

## Requirements

1. **Multiple Teams and Players**: The game must support various teams with several players.
2. **Unique Identifiers**: Each game, team, and player must have a unique identifier and name.
3. **Single Instance**: Only one game instance can exist in memory at any time.
4. **Name Uniqueness**: Ensure that game and team names are unique and check for name conflicts during creation.

## [Design Constraints](#_2et92p0)

1. **Web-Based Environment**: The game application must be compatible with various web browsers and operate efficiently across different platforms (e.g., Windows, Mac, Linux, mobile devices). This requires a responsive design and cross-platform testing.
2. **Single Instance Management**: The singleton design pattern must be implemented to ensure only one instance of the game is active, which can complicate the management of game states and instance interactions.
3. **Unique Identifiers**: Generating unique identifiers for each entity can involve complex algorithms or databases to avoid collisions, particularly in a distributed environment.
4. **Concurrency**: The application must handle simultaneous users and interactions without performance degradation, necessitating robust error handling and synchronization mechanisms.

## [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)

The UML class diagram for the game application includes the following classes:

* **Entity**: A base class with common attributes 'id' and 'name' used by Game, Team, and Player classes.
* **Game**: Inherits from Entity and manages the overall game state, including teams and rounds.
* **Team**: Inherits from Entity and contains a list of Player instances.
* **Player**: Inherits from Entity and represents individual players within a team.

**Object-Oriented Programming Principles**:

* **Inheritance**: The Game, Team, and Player classes inherit from the Entity class, promoting code reuse and reducing redundancy.
* **Encapsulation**: Each class encapsulates its attributes and behaviors, ensuring that the internal state is protected from outside interference.
* **Singleton Pattern**: The GameService class implements the singleton pattern to ensure only one instance of the game service exists, providing global access to game management functionalities.
* **Iterator Pattern**: Used in managing collections of games, teams, and players, allowing for efficient iteration and management of these entities.

**"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.

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 applications, but it is less commonly used for large-scale web server deployment. Higher hardware costs and limited configuration flexibility make it less ideal for server-side hosting. | Linux is a preferred choice for server-side hosting due to its stability, security, and open-source nature. It is cost-effective, supports scalability, and is highly configurable. | Windows Server is widely used in enterprise settings. It integrates well with Microsoft technologies, but licensing fees and maintenance can be costly compared to Linux. | Mobile devices do not function as servers but can serve as clients that interact with the web-based application hosted on a server. |
| **Client Side** | Mac development offers excellent compatibility for iOS devices using tools like Xcode. However, ensuring cross-platform compatibility across other desktop operating systems requires additional testing. | Linux can be used for web-based client applications, but requires specialized knowledge for setup. It’s ideal for low-cost development and testing but less commonly used for desktop clients. | Windows is commonly used for client-side development due to its extensive support for development tools and easy integration with .NET-based systems. It is highly compatible with most desktop browsers. | Mobile devices require a responsive design to ensure compatibility with both iOS and Android systems. Additional testing is necessary for mobile browser performance and touch interface interactions. |
| **Development Tools** | Tools like Xcode (for iOS), IntelliJ IDEA, and Visual Studio Code are commonly used. Mac is necessary for iOS app development and testing, but hardware costs are high. | Eclipse, IntelliJ IDEA, and Visual Studio Code are widely used for Linux development. Open-source tools keep costs low, but Linux expertise is essential. | Visual Studio, IntelliJ IDEA, and Eclipse are widely used on Windows. Strong support for .NET-based development. Windows licensing costs for development tools may apply. | Android Studio (for Android) and Xcode (for iOS) are essential for mobile development. Web-based development tools are required for cross-platform compatibility across mobile devices and web browsers. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* Operating Platform: I recommend using Linux as the server operating platform to expand Draw It or Lose It to various computing environments. Linux is widely recognized for its stability, security, and scalability, making it suitable for hosting multi-user applications. Its open-source nature also allows for customization and cost-effectiveness, essential for a growing gaming application.
  + Operating System Architectures: Linux supports several architectures, including:
    - Monolithic Kernel Architecture: In this architecture, the entire operating system runs as a single program in kernel space, allowing for high performance and efficient communication between components. This is beneficial for real-time game performance.
    - Microkernel Architecture: Although not typical for Linux, some distributions allow for microkernel implementations that run only the most essential services in kernel space, reducing system crashes and enhancing security.
    - Client-Server Model: This architecture allows multiple clients (users) to connect to a central server where game logic is processed, which is ideal for a multi-user game application.

These architectures enable efficient resource management, which is essential for running multiple game instances while ensuring responsiveness.

* Storage Management: For storage management, I recommend using PostgreSQL or MySQL as the relational database management system (RDBMS) for storing user data, game states, and the image library. Both databases offer:
* Robust Data Management: They provide ACID compliance for transactions, ensuring that user data and game states are accurately managed and preserved.
* Scalability: As the game grows and user numbers increase, these databases can scale horizontally to accommodate additional loads without significant performance degradation.
* Efficient Query Processing: Complex queries for retrieving game data can be efficiently executed, enhancing user experience during gameplay.
* Memory Management: Linux employs several memory management techniques that will benefit the Draw It or Lose It application:
* Virtual Memory: Linux uses virtual memory to manage the game's memory resources efficiently. This allows the application to use more memory than is physically available by swapping pages in and out of disk storage.
* Memory Paging: The system can quickly allocate memory pages for each game instance, ensuring multiple users can run their games simultaneously without losing performance.
* Memory Caching: Frequently accessed images and game data can be cached in memory to minimize latency, improving the game's responsiveness.
* Distributed Systems and Networks: I recommend implementing a RESTful API for the Draw It or Lose It application to facilitate communication between various platforms. This allows for seamless interactions between the client-side applications (running on various platforms) and the server. Key points include:
* Decoupling of Components: A distributed architecture allows different components (frontend clients and backend services) to operate independently, enhancing system robustness.
* Network Connectivity: Leveraging cloud-based solutions ensures reliable connectivity, enabling users to access the game from anywhere. Implementing load balancers can manage traffic and ensure high availability, even during peak usage times.
* Handling Outages: The distributed system should be designed to gracefully handle connectivity issues, with fallback mechanisms such as caching recent game states locally to allow users to continue their gameplay in case of temporary outages.
* Security: To protect user information across various platforms, the following security measures should be implemented:
* Encryption: Use SSL/TLS for secure communication between clients and the server to protect data in transit. Data at rest in the database should also be encrypted to prevent unauthorized access.
* Multi-Factor Authentication (MFA): Implement MFA to enhance user account security, requiring users to provide additional verification beyond passwords.
* Regular Security Audits: Conduct routine audits and vulnerability assessments to identify and address potential security risks promptly. This can include employing tools like OWASP ZAP for automated testing.

By adopting these recommendations, Draw It or Lose It will be well-positioned to successfully expand across multiple operating platforms while ensuring efficient performance, robust data management, and strong security.