

<Draw-It-or-Lose-It >

# **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 | <07/20/24> | |  | | --- | |  |  |  | | --- | | Victor Tran | | |  | | --- | |  |  |  | | --- | | Initial version of the software design document | |
| 1.5 | <08/03/24> | Victor Tran | Changed Evaluation Section and Requirement |
| 2.0 | <08/19/24> | Victor Tran | Improved Requirement section and finalized |

**Instructions**

The Gaming Room aims to expand its existing Android game, "Draw It or Lose It," to multiple platforms, including web and mobile environments. This expansion requires a software design that ensures the game can handle multiple teams, each with multiple players, while maintaining unique game, team, and player names. The design must also enforce that only one instance of the game can exist in memory at any given time. To meet these needs, the solution leverages design patterns such as Singleton and Iterator to manage the game instances and enforce uniqueness.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to expand its existing Android game, "Draw It or Lose It," to multiple platforms, including web and mobile environments. This expansion requires a software design that ensures the game can handle multiple teams, each with multiple players, while maintaining unique game, team, and player names. The design must also enforce that only one instance of the game can exist in memory at any given time. To meet these needs, the solution leverages design patterns such as Singleton and Iterator to manage the game instances and enforce uniqueness.

## Requirements

***Business Needs:***

* *The game must be accessible on multiple platforms, including the web and mobile devices.*
* *The system must ensure that game, team, and player names are unique.*

***Technical Needs:***

* *Only one instance of the game should run at a time to avoid conflicts.*
* *The application must manage multiple teams and players within a game, ensuring no name is reused.*

## [Design Constraints](#_2et92p0)

Developing the game in a web-based distributed environment presents several constraints:

* **Performance:** The game must perform well across different platforms, handling multiple concurrent users without slowing down.
* **Security:** The application must protect user data and maintain the integrity of game instances.
* **Scalability:** The design should support future expansions, such as adding more features or supporting additional platforms, without requiring significant rework.
* **Maintainability:** The code must be easy to update and maintain, allowing for quick bug fixes and feature additions.

## [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 represents the core components of the game application:

* **Game:** Manages a list of teams. It is responsible for adding teams and ensuring that team names are unique.
* **Team:** Manages a list of players. It is responsible for adding players and ensuring that player names are unique.
* **Player:** Represents a participant in the game.
* **GameService:** Implements the Singleton pattern to ensure only one instance of the game service exists. It manages multiple games, each with unique names.

Object-oriented programming principles such as inheritance (in scenarios where classes share common attributes), encapsulation (where each class manages its data and methods), and composition (where Game is composed of Teams, and Teams are composed of Players) are used to efficiently manage and organize the software's structure.

**"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** | Macs are well-regarded for their robust development environments, especially in creative industries. However, their high cost and lower adoption rate in server environments make them less ideal for hosting large-scale web applications. While macOS offers a stable and secure environment, it is better suited for development rather than production server hosting. Therefore, a Mac could be used effectively for testing and development purposes, but it is not recommended for hosting a web-based game application in a production environment. | Linux is the preferred platform for hosting web-based applications due to its open-source nature, stability, and security. It is highly reliable in handling server loads and offers flexibility in configuration. The cost-effectiveness of Linux, being free and open-source, allows for scaling without the burden of licensing fees. Linux distributions like Ubuntu Server or CentOS provide extensive community and enterprise support, making them ideal for deploying large-scale, high-availability systems. | Windows servers are widely used in enterprise environments and offer extensive support for various enterprise applications. However, they come with significant licensing costs, especially for large-scale deployments. While Windows provides a familiar environment for those accustomed to Microsoft products, it demands more resources and frequent updates to maintain security and performance. Despite these challenges, Windows servers remain a viable option, particularly for organizations already invested in Microsoft’s ecosystem. | Mobile devices, due to their limited processing power and storage, are unsuitable for hosting server applications. However, they are critical client-side devices designed to connect to web servers and facilitate user interactions. The server-side infrastructure must ensure robust and reliable connections for mobile clients, guaranteeing smooth and responsive gameplay. |
| **Client Side** | Macs are popular among developers and users in creative and educational sectors. Ensuring compatibility with macOS is important, particularly with Safari and Retina displays. Although Macs represent a smaller market share compared to Windows, optimizing the game for macOS ensures a broader audience reach, particularly among the niche markets that favor Apple products. | Linux, while powerful and flexible, is less commonly used as a primary OS by end users. Ensuring compatibility with major browsers like Firefox and Chrome is crucial for Linux users. Testing the game on popular Linux distributions ensures that it runs smoothly, even if the overall user base is smaller compared to other operating systems. | Windows is the most widely used operating system, making it critical for the game to run smoothly on Windows to reach the largest possible audience. Ensuring compatibility across different versions of Windows (such as Windows 10 and 11) and supporting popular browsers like Chrome, Edge, and Firefox is essential. Robust performance on Windows will be a key factor in the game’s success. | Mobile devices are essential for the game’s success, as many users prefer playing games on smartphones or tablets. Ensuring compatibility with both Android and iOS platforms is crucial, as these two operating systems dominate the mobile market. The game should be optimized for various screen sizes and touch interfaces, providing a consistent and responsive experience. Additionally, mobile versions of the game should be lightweight to ensure quick load times and low data usage, catering to users with varying network conditions. |
| **Development Tools** | Mac offers robust development tools such as Xcode, which is essential for iOS app development. Additionally, popular IDEs like IntelliJ IDEA and Eclipse are supported, making Mac a strong choice for developers working in multiple programming languages. The development environment on Mac is well-suited for creating high-quality software, especially within Apple’s ecosystem. However, the higher cost of Mac hardware may be a consideration for development teams, especially those on a tighter budget. | Linux supports a wide range of open-source development tools, including Eclipse, VS Code, and NetBeans. Developers who prefer an open-source environment appreciate the flexibility Linux offers. However, Linux may have a steeper learning curve, requiring more expertise and time to configure the development environment to meet specific needs. Despite this, the vast range of available tools and the cost savings of open-source software make Linux an attractive option for development. | Windows provides a broad selection of development tools, with Visual Studio being one of the most powerful and popular IDEs available. Windows also supports various other IDEs like IntelliJ IDEA, Eclipse, and PyCharm. The development environment is user-friendly, with extensive support and documentation. However, licensing costs can be a concern, particularly when using enterprise-level software. Despite this, Windows is well-suited for developing applications intended for a wide user base due to its large market share. | Mobile development requires specific tools like Android Studio for Android and Xcode for iOS. Familiarity with these tools is crucial for building and testing the game on mobile platforms. The development process must account for variations in hardware, screen sizes, and OS versions to ensure a consistent user experience across devices. These tools provide comprehensive environments for developing, debugging, and deploying mobile applications. |

## 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**: For the expansion of The Gaming Room's "Draw It or Lose It" game, I recommend using **Linux** as the primary operating platform. Linux is renowned for its stability, security, and scalability, making it ideal for hosting web-based applications that require high availability and performance. Its open-source nature also means that The Gaming Room can scale operations without worrying about significant licensing costs, which is a key consideration as the game grows in popularity.
2. **Operating Systems Architectures:** The recommended architecture for "Draw It or Lose It" is a **client-server architecture**. In this setup, the server—running on a reliable Linux distribution like Ubuntu Server or CentOS—will handle critical operations such as game management, data storage, and user authentication. Players will connect to this server via web browsers or mobile apps, enabling cross-platform play. To manage and scale the server infrastructure efficiently, tools like Docker can be employed, allowing seamless scaling and maintenance as the number of active players increases.
3. **Storage Management**: For data storage, I recommend leveraging **cloud storage services** such as Amazon S3 or Google Cloud Storage. These services offer robust, scalable, and secure storage solutions that can easily handle the large amounts of data generated by the game, including player information, game progress, and multimedia assets. Additionally, these cloud services provide features like automated backups and global accessibility, ensuring that the game data remains safe and readily available, even in the event of hardware failures.
4. **Memory Management**: Linux's **efficient memory management** is a significant advantage in this scenario. The operating system is adept at managing memory allocation dynamically, ensuring that the server can support multiple concurrent players without experiencing slowdowns. Linux intelligently prioritizes active processes and reclaims memory from idle processes, which is crucial for maintaining a smooth gaming experience even during peak usage periods.
5. **Distributed Systems and Networks**: To facilitate **cross-platform play** and ensure that the game remains responsive regardless of the players' locations, I suggest implementing a distributed system. This system would involve multiple servers working in tandem, using APIs to communicate and share game data efficiently. The network should be optimized to minimize latency, which is critical for a real-time game like "Draw It or Lose It." Additionally, implementing redundant servers will help maintain game availability even if one server encounters issues, ensuring a consistent and reliable gaming experience.
6. **Security**: Given the importance of protecting player data, I recommend a strong focus on **security**. This includes using encryption to safeguard data both at rest and in transit, as well as leveraging Linux's built-in security features, such as firewalls and regular security updates. Access control mechanisms should be implemented to restrict sensitive operations to authorized users only. Furthermore, monitoring tools should be used to detect and respond to any potential security threats in real-time, maintaining the integrity and trustworthiness of the game platform.