

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc152056589)

[Table of Contents 2](#_Toc152056590)

[Document Revision History 2](#_Toc152056591)

[Executive Summary 3](#_Toc152056592)

[Requirements 3](#_Toc152056593)

[Design Constraints 3](#_Toc152056594)

[System Architecture View 4](#_Toc152056595)

[Domain Model 4](#_Toc152056596)

[Evaluation 5](#_Toc152056597)

[Recommendations 8](#_Toc152056598)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/2023 | Gwen Magyar | Updated document |
| 1.1 | 11/25/2023 | Gwen Magyar | Added evaluation and recommendation |

## [Executive Summary](#_sbfa50wo7nsh)

The challenge in software design involves crafting a game application within a web-based distributed environment. Our suggested resolution entails constructing a Java application, employing object-oriented programming principles. This application will feature key classes such as GameService, Game, Team, and Player, fostering interaction for efficient management of games, teams, and players. The design will capitalize on the Entity class, serving as a universal base for all entities, with shared attributes such as ID and name. Embracing this design methodology allows us to effectively meet software requirements, resulting in the development of a fully functional game application.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

These design constraints collectively shape the application development process, requiring meticulous consideration of network communication, security measures, Java-based technologies, object-oriented design principles, scalability, performance optimization, and user interface design.

Web-Based Distributed Environment: Our game application necessitates development within a web-based distributed environment, signifying its accessibility over the internet and support for multiple concurrent users. This constraint mandates a design that can gracefully handle network latency and address security considerations.

Java Programming Language: The software's development is mandated to utilize the Java programming language. This constraint confines our technology stack to Java-based frameworks, libraries, and tools, demanding adherence to Java coding conventions and best practices.

Object-Oriented Design: Adhering to object-oriented design principles is paramount for fostering modularity, reusability, and maintainability. This constraint entails crafting classes with proper encapsulation, inheritance, and polymorphism, along with the application of design patterns and abstraction techniques to establish a flexible and extensible codebase.

Scalability and Performance: The game application's design must anticipate and accommodate growth in games, teams, and players. Ensuring scalability to handle an expanding user base without compromising performance is crucial. This constraint mandates optimizing database access, minimizing resource consumption, and implementing caching mechanisms.

User Interface: The user interface of our application must be intuitive, user-friendly, and responsive, enhancing the overall player experience. This constraint calls for the design of a visually appealing and responsive interface using web technologies like HTML, CSS, and JavaScript.

## [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 serves as a visual representation of the game application's domain model, comprising essential classes such as GameService, Game, Team, and Player. These classes are intricately connected to fulfill the underlying software requirements.

Within this model, the Entity class plays a pivotal role as the foundational class for all entities in the application, providing common attributes like ID and name. Embracing the principle of inheritance, this class allows other entities to inherit these attributes and behaviors, ensuring consistency across the application and minimizing code duplication.

The GameService class assumes a central role as the primary service for managing games. It oversees a list of games and offers methods for adding games, retrieving games by ID or name, and obtaining the total game count. Following the Singleton design pattern, this class guarantees a singular, accessible instance throughout the application. The SingletonTester class uses the ProgramDriver class to effectively test the singleton behavior.

In the broader context, the Game class embodies a game within the application, maintaining a list of participating teams and providing methods for adding teams and retrieving a string representation of the game. The Team class, in turn, represents a team within a game, managing a roster of players and offering methods for adding players and retrieving a string representation of the team. Finally, the Player class encapsulates an individual player, featuring a method to retrieve a string representation.

These classes collectively exemplify the principle of composition, where intricate objects are constructed by combining simpler ones. In this context, the Game class comprises teams, and each Team comprises players. This compositional approach establishes a hierarchical structure that efficiently manages games, teams, and players within the application.

**"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** | Mac is known for its robust Unix-based architecture, making it a reliable choice for hosting web-based software applications. Its closed ecosystem provides a stable environment, and the macOS Server offers features like Apache and Nginx for web hosting. However, limited hardware options and higher costs can be drawbacks for large-scale server deployments | Linux excels in server-side applications due to its open-source nature, scalability, and versatility. It supports a wide range of server configurations, offers strong security features, and is highly customizable. The lack of a unified user interface may be challenging for some users, but its cost-effectiveness and stability make it a preferred choice for hosting web applications. | Windows servers are widely used in enterprise environments, especially when compatibility with Microsoft technologies is crucial. They provide a user-friendly interface, extensive support for various applications, and compatibility with ASP.NET. However, licensing costs and susceptibility to security threats may be concerns, and it may not be as suitable for hosting certain open-source technologies. | Mobile devices are not typically used as server platforms for hosting web-based software applications due to limitations in processing power, storage, and continuous availability. While some mobile devices support server-like functionalities for specific applications, they are not designed to be primary hosts for web applications due to their inherent constraints. |
| **Client Side** | Supporting multiple types of clients on Mac involves considerations such as higher development costs, a need for expertise in macOS-specific technologies, and potential challenges in cross-platform compatibility. However, the affluent user base and the popularity of macOS among creative professionals can justify the investment in certain applications. | Developing software for multiple clients on Linux requires expertise in open-source technologies. While development costs may be lower, ensuring compatibility across different Linux distributions can be challenging. Linux users often expect open-source solutions, making it essential to align development strategies with the Linux community's ethos. | Supporting clients on Windows involves considerations for a vast user base. Developing software for Windows may require expertise in .NET technologies and adherence to Windows-specific design principles. The cost of development can vary, and time-to-market may be influenced by the need for compatibility testing across different Windows versions. | Developing software for various mobile devices necessitates expertise in mobile app development frameworks. Costs can vary based on the platforms targeted, with considerations for iOS and Android having distinct requirements. Time-to-market is crucial in the fast-paced mobile ecosystem, requiring efficient development processes and ongoing updates for evolving mobile standards. |
| **Development Tools** | Swift and Objective-C are the common languages used. Xcode is used as Apple’s IDE. | C, C++, Python, and Java are commonly used languages. IDEs include Visual Studio Code, Eclipse, and PyCharm. | C, C++, Python, C#, .Net, and Java are commonly used languages. IDEs include Visual Studio Code, Eclipse, and PyCharm. | Swift and Objective-C are the common languages used for Apple devices with Xcode used as the IDE. For Android, Java and Kotlin are the commonly used languages with Android Studio as the IDE. |

## Recommendations

Analyzing the characteristics of various systems architectures, I recommend the following for The Gaming Room's expansion of Draw It or Lose It:

1. **Operating Platform Recommendation:** Recommend Linux as the operating platform to facilitate the expansion of Draw It or Lose It to other computing environments. Linux's cost-effectiveness, scalability, and robust developer support make it an ideal choice for accommodating diverse computing environments.
2. **Operating Systems Architectures:** Linux operates on a Unix-based architecture, known for its stability and versatility. It supports a multi-user environment and utilizes a monolithic kernel architecture, ensuring efficient resource utilization and robust system performance. The open-source nature of Linux enables customization to meet specific requirements.
3. **Storage Management:** Utilize a distributed storage management system compatible with Linux, such as the Ceph distributed storage system. Ceph offers scalability and fault tolerance, crucial for a growing application like Draw It or Lose It. Its ability to manage object, block, and file storage makes it suitable for diverse storage needs.
4. **Memory Management:** Linux employs advanced memory management techniques for optimal performance of software applications. Draw It or Lose It can benefit from Linux's virtual memory system, which efficiently manages RAM resources by employing techniques like demand paging and memory mapping.
5. **Distributed Systems and Networks:** To enable communication between various platforms, implement a microservices architecture for Draw It or Lose It. This approach allows for the development and deployment of loosely coupled, independently scalable components. Utilize RESTful APIs for communication between services, ensuring interoperability across different platforms. Consider redundancy and fault tolerance to address connectivity issues and outages within the distributed system.
6. **Security:** Linux offers robust security features that align with The Gaming Room's emphasis on user protection. Implement secure communication protocols such as HTTPS for data transfer between platforms. Utilize Linux's built-in security mechanisms, including access controls and user permissions, to safeguard user information. Regular security audits and updates are essential to address emerging threats and vulnerabilities in the operating platform.

Leveraging Linux's strengths in scalability, open-source adaptability, and robust security features will provide a solid foundation for expanding Draw It or Lose It across diverse computing environments. Incorporating distributed systems and network strategies will enhance communication between platforms while ensuring the security and integrity of user information.