

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.3

## 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 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/16/2023 | Jorge Cisneros Rodriguez | Initial version and proposal of the design template |
| 1.2 | 07/30/2023 | Jorge Cisneros Rodriguez | Additional changes to Evaluation |
| 1.3 | 08/12/2023 | Jorge Cisneros Rodriguez | Additional revisions to Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The task involves transforming an Android game, Draw It or Lose It, into a web-based multi-platform application. The Gaming Room requires a comprehensive design strategy that meets the unique requirements of the web-based version of the game. The proposed solution involves developing the game using Java and leveraging software design patterns like the Singleton and Iterator patterns. The web-based game will uphold the original game mechanics while introducing unique game and team names and optimal memory management through Singleton design pattern.

## Requirements

The client's requirements are to create a web-based version of the game Draw It or Lose It. The game should maintain the original game mechanics, with teams competing to guess the meaning of rendered images. The software should allow for multiple teams and players, with unique game and team names. Only one instance of the game should exist in memory at any given time. The software design should ensure optimal memory management, and the code should be clear and maintainable, following industry-standard best practices.

## [Design Constraints](#_2et92p0)

1. The game needs to be a web-based application that can run on multiple platforms. This requires the design to be adaptable to different browsers and devices, which can introduce complexities in maintaining a consistent user experience.
2. The Singleton design pattern will be implemented to ensure that only one instance of the game exists in memory at any one time. This can introduce challenges in multithreaded environments, but we will use appropriate synchronization techniques to handle this constraint.
3. Team and game names must be unique. This introduces the constraint of having to check and validate the uniqueness of names, which could potentially impact response times.
4. As the application grows to accommodate more players, teams, and games, there could be scalability issues. These constraints will be mitigated through proper architectural design and efficient use of data structures.

## [System Architecture View](#_ilbxbyevv6b6)

The game application will be developed using a three-tier architecture model, which includes the client, application server, and database layers. The client layer comprises the user interface, the application server hosts the game logic, and the database layer manages the game's data.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram depicts the structure of the game application, which consists of seven classes: Entity, GameService, Game, Team, Player, ProgramDriver, and SingletonTester. Entity is the base class that provides common attributes and behaviors to the Game, Team, and Player classes through inheritance. GameService is a singleton class that manages all the instances of the Game class and uses iterator patterns for game name uniqueness checks. Each Game instance can contain multiple Team instances, and each Team instance can have multiple Player instances, demonstrating the principle of aggregation.**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac's server-side architecture is stable, secure, and suitable for hosting the game application. However, costs may be higher due to the proprietary nature of Apple's ecosystem. | Linux is highly viable for server-side hosting of the game application. Its stability, security, open-source nature, and a variety of deployment methods make it cost-effective and optimal. | Windows server environment is user-friendly and can efficiently host the game application. It may not be as cost-effective as Linux for large-scale deployments but offers reliable deployment methods. | Servers would not typically be hosted on mobile devices. Mobile clients can access the game hosted on servers, with deployment managed through cloud services. |
| **Client Side** | Developing a responsive HTML interface for Mac clients requires expertise in Java and appropriate IDEs. The cost and time for development could be higher due to the specific requirements of Apple's ecosystem, but standard web technologies ensure browser compatibility. | For Linux clients, expertise in Java is necessary. The development process could be less costly and quicker due to the open-source nature of many Linux tools and environments. Standard web technologies ensure browser compatibility. | Windows has a large user base, making it an important platform for client-side game deployment. Developing a responsive HTML interface for Windows may require moderate cost and time investment. Standard web technologies ensure browser compatibility. | For mobile devices, a version of the game could be developed in Java for Android and Swift or Objective-C for iOS, potentially requiring two separate development teams. |
| **Development Tools** | For Mac, Java development would typically use IntelliJ IDEA or Eclipse, Maven for dependency management, and Git for version control. Some tools may have licensing costs. Mac development may require a separate team due to specific expertise and higher development costs. | On Linux, Java can be written and compiled using a variety of tools, including IntelliJ IDEA, Eclipse, and NetBeans. Git could be used for version control. Linux supports a variety of open-source development tools with no licensing costs, and Linux development could be quicker and less costly. | For Windows, IntelliJ IDEA or Eclipse would be the go-to choice for Java development. Additional tools may include Maven for dependency management and Git for version control. Windows development requires a dedicated team. | Mobile development would use Android Studio for Android and Xcode for iOS. Potential licensing costs include the annual fee for Apple's Developer Program. Mobile development requires specialized teams, increasing costs and complexity. |

## 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:** Based on a comprehensive analysis of different operating systems and their respective attributes, I recommend adopting a Linux-based operating system for the Draw It or Lose It game's server platform. Linux's inherent stability, customizability, and cost-effectiveness, coupled with its vast community support, make it a frontrunner. Moreover, the open-source nature of Linux provides opportunities for tailoring it specifically to the game's requirements.
2. **Operating Systems Architectures**: Linux offers modular monolithic kernel architecture, allowing components to be dynamically loaded or unloaded as necessary. This flexibility ensures that the server can be optimized for specific tasks, enhancing the overall performance of the game. Moreover, the multi-user and multitasking capabilities native to Linux provide a seamless experience to a large number of users concurrently accessing the game.
3. **Storage Management**: For the game's data storage needs, MySQL is recommended. MySQL is renowned for its speed, reliability, and ease of use. It is an open-source relational database management system and has extensive support for web-based applications. Moreover, it integrates well with Linux, further streamlining the deployment and maintenance process.
4. **Memory Management**: Linux boasts a sophisticated memory management system, ensuring optimal performance for the game. Virtual memory in Linux allows for larger processes to run efficiently, even with limited physical RAM. Furthermore, the Swap Space feature ensures that inactive pages are moved to disk space, thus freeing up RAM for more pressing tasks. This dynamic memory allocation and de-allocation mechanism will ensure that the game performs smoothly even under heavy loads.
5. **Distributed Systems and Networks**: To accommodate the anticipated growth of the game and to ensure resilience and high availability, it's essential to deploy the game in a distributed environment. Docker offers containerization, ensuring that the game runs consistently across different computing environments. Kubernetes can be utilized to manage and orchestrate these containers, ensuring automatic scaling, failover, and more. To guarantee a stable connection among various platforms, the game can be interconnected via the robust TCP/IP protocol, which ensures reliable data transfer.
6. **Security**: Ensuring the security of user data and maintaining a secure communication channel is crucial. To achieve this, implementing HTTPS is vital, as it ensures encrypted communication between the client and the server. For user data protection, a combination of encryption techniques, secure hash algorithms, and salted passwords is recommended. Linux also offers various firewall and intrusion detection systems, further fortifying the game against potential threats. Regular security audits, vulnerability assessments, and adherence to best practices will solidify the game's defense against cyber threats.