

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_u6u4ltdz2gfr)

[**Table of Contents 2**](#_i8iy9kg0qlm8)

[**Document Revision History 2**](#_t2in291un3r2)

[**Executive Summary 3**](#_dqme1y107xbp)

[**Requirements 3**](#_hxmzvk4ftm0k)

[**Design Constraints 3**](#_ff2s09kv3ikp)

[**System Architecture View 3**](#_zd92c2oetxp4)

[**Domain Model 3**](#_crfjymu0n0xj)

[**Evaluation 4**](#_r2xb25m8dnpd)

[**Recommendations 5**](#_u5x077yaxkhq)

## [Document Revision History](#_9g0pfjxfj7ge)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/20/2025 | Brandon Fluegge | Updates to Executive Summary and Design Constraints |
| 1.1 | 08/01/2025 | Brandon Fluegge | Updates to Evaluation |

**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](#_5rntaajarr91)

The software design problem presented is establishing a development environment for the game, Draw it or Lose it, as a web-based application. The solution to this problem is to establish a singleton pattern to address the following: the ability to have one or more teams involved, each team having multiple players assigned to it, game and team names should be unique, only one instance of the game can exist in memory. All of these can be addressed by allowing only one instance of a player, team, and game id to exist within the game at any point in time.

## Requirements

* *Java development environment*
* *Servers for hosting web-based application*

## [Design Constraints](#_nhuqp6tqazlo)

The design constraints of this project include, but are not limited to, the hosting environment limiting scalability, performance, and resources, security measures must be robust in a web-based application, the app must function across varying web browsers, the app must function across varying devices, and platform guidelines must be adhered to.

## [System Architecture View](#_oj5kgg412rhk)

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](#_to4wgpscit0b)

The UML class diagram below demonstrates a singleton pattern where the GameService object created can be accessed from anywhere in the program, but the instance created is protected from being overwritten. The GameService class has a private constructor that prevents other classes from accessing and creating new instances of the entity object. This ensures there will only be one instance of GameService created. This object is accessed through the getInstance() method. Using this method allows multiple unique team, game, and player ID’s to be created without altering the original GameService object.

**"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](#_86nvlsv2zxyq)

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** | In hosting a server on a Mac, there are several things to consider. For example, Mac offers strong security and Apple device integration. However, Mac also comes with high cost, limited customization features, and limited scalability. All of this being said, Mac is a suitable, stable choice for server side development. | Hosting a server on Linux is the optimal choice. Linux is open-source, cost effective, stable, and secure. Some minor issues a developer may encounter are issues with software compatibility, limited commercial support, and hardware compatibility issues. However, all of these can be addressed relatively simply, and there is a large community of users out there who can tell you how. | Windows is a good option for server side development. It is user friendly, is compatible with most (if not all) software, and comes with a very large amount of support. The downsides to Windows are what make it a less optimal choice than Linux and those include: cost, security, and resources. Windows is a commercial product which leads to higher costs. It is widely used, causing quite a few security issues. Also, Windows is very resource intensive which takes away from the server being hosted. | I do not agree with using mobile devices for application development or server side hosting because of the lack of performance, the lack of security, and the need for network dependency. Although these devices have a wide reach and are relatively low-cost, they are not efficient server development devices. |
| **Client Side** | Mac is a very viable option for client side development because it is stable and developer friendly. There may be some setbacks with software availability and the high cost of an Apple computer. | Linux is a cost effective client side programming platform, but it comes with a learning curve. Programmers must spend time learning and getting familiar with the OS. There are plenty of forums and documentation for conquering such a task. | Windows has a very large user base and comes with robust technical support. However, a Windows machine is often very resource intensive and can often lead to required upgrades in hardware for larger demand of processes. | Development for client side programs on mobile devices is time consuming and testing the program is an issue. Your development environment may need to be configured to properly test on these devices. However, mobile devices require limited maintenance and are very common. |
| **Development Tools** | * Swift, C, Java * Sublime text, Visual Studio (or equivalent IDE) * Git * Safari web browser | * Java, C, Python * Sublime text (or equivalent IDE) * Git * Web browsers (firefox, chrome, etc) | * Java, C, Python * Visual Studio, VS Code (or equivalent IDE) * Git * Web browsers | * Kotlin, Swift, Java * Android SDK, Xcode * Git * Safari (or equivalent mobile web browser) |

## 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**: Linux is the most stable, cost-effective, secure operating environment. As a customizable option for hosting the server of Draw it or Lose it, Linux offers the most versatility for developers. Also, although Linux may lack commercial support, the operating system makes up for it in a large, active community and extensive documentation.
2. **Operating Systems Architectures**: A multi-tiered architecture is the most optimal choice for Draw it or Lose it. This is because multi-tier architectures allow for scaling by separating the application into tiers.
3. **Storage Management**: It is recommended that the server be equipped with a solid state drive to handle the storage load of Draw it or Lose it. This recommendation is based on the speed that SSD provides in comparison to HDD. Utilizing solid state drives will improve the performance of the game.
4. **Memory Management**: Linux utilizes a paging system for resource optimization. Paging systems work by separating data into pages and storing data according to its usage. This method allows projects more memory than what is physically available by utilizing virtual memory. This provides efficiency by storing data that is not being used directly to the storage device. Freeing up memory for current processes and operations
5. **Distributed Systems and Networks**: Restful APIs are the optimal solution here. These APIs provide communication through the hyper text transfer protocol and are often used for web app development.
6. **Security**: Linux provides robust security because it is open source and customizable. It is important to keep the system and hardware updated, require strong passwords from users, perform sanity checks on the server side of the game, and adhere to least privilege principles.