

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 8/16/2025 | Nicholas Deniz | Edited the recommendations |
| 2.0 | 8/2/2025 | Nicholas Deniz | Edited the Evaluation section |
| 1.0 | 07/19/25 | Nicholas Deniz | Filling in the Executive Summary, their requirements, the Design Constraints, and their Rationale, describing the UML diagram, evaluating operating platforms, and creating recommendations for The Gaming Room. |

**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)

Draw It or Lose It is a game where teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit. The client, The Gaming Room, wants to develop a web-based game that serves multiple platforms based on their current game. It is currently an Android only app, but we will make sure it runs on multiple platforms like being able to play on different browsers like Safari and Firefox. In Java we will operate the game, teams, and players under the guidelines given with the use of a singleton and iterators:

* 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.

## Requirements

Business Requirements:

* The game will be multi-platform, so it will run on different systems.
* Keep to the guidelines given about the game rules.
* Game and team names must be unique.
* Able to support more than one team and each team has multiple players.

Technical Requirements:

* Use a singleton so there is only one instance.
* Use an iterator to make sure of names and Id being unique.
* Has the capacity to have multiple teams with multiple players

## [Design Constraints](#_2et92p0)

- Use Singleton; only one instance is allowed and must prevent instantiations. GameService will have the getInstance() method.

Implication: Make sure all game data is centered to avoid issues.

- Use Iterators; runs through a list of names to make sure there are no duplicates.

Implication: This is to make each name and Id unique for the game, team, and player to avoid conflicts with having the same identification.

- Must run on multiple platforms; be able to run in different browsers like Safari and Firefox.

Implications: Have it been able to reach a larger audience.

- Be able to hold multiple teams with multiple players in each team.

Implication: Allow more players and make the game more enjoyable.

-Must run on Java for all the code.

Implication: It will avoid conflicts if it is all coded the same and make it easier to integrate into a web-based environment.

## [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)

In the UML class diagram, ProgramDriver, the main(), uses SingletonTester, so it is used to verify the GameService.getInstance() is the same. Game service holds the single instance and has a list for Game checks the Id’s. It also has many methods like addGame to check the games name and create a new one if it doesn’t. It also has getGame(id)/getGame(name) to look it up and getGameCount() to see how many there are. The other important one is getNextTeamId()/getNextPlayerId() to get those unique titles. Entity is the base class holding Id and name which will be inherited across classes. Game inherits Entity; Game holds a list of Team objects. Team inherits Entity; Team holds a list of Player objects. Player inherits Entity; it is empty. GameService starts a line to Game to Team to Player with 0…\* to shows that each class that the line goes to can have nothing, a few or many. For example, Team can have no players, a few players, or many players.

**"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** | There are pros on the server side of Mac. It has a built in Apache server which works with HTML and JavaScript well. It also supports Nginx through Homebrew so that frameworks are easier to work with like React. The macOS licensing is included with hardware. There are cons on the server side of Mac. It is not used commonly among the community. Mac isn’t specifically made to support it at a large scale. There would be trouble operating it with heavy amounts of people using it. | There are pros on the server side of Linux. It is commonly used for web-based software applications. It is open sourced so there is possibility for customization  It is compatible with a lot like Apache, Python, SQL, Nginx, and Node.js. It is powerful overall due to its Command-Line interface. There is also no OS license cost. There are cons on the server side of Linux. It is tough to understand and use. It isn’t usually the best for games due to the graphics being limited and it is not able to handle them well. There is also common hardware that may not be supported. | There are pros on the server side of Windows. It supports Microsoft tech like MS SQL Server and .NET Core. It has a GUI control panel for easier management. Different from Linux. There are regular security updates. It can handle a heavy load of people. There are cons on the server side of Windows. It is less flexible than Linux. There is not much open-source support for things like MySQL. It requires more resources like CPU and memory. There will be a high licensing cost involved. | There are pros to the server side of Mobile Devices. It is portability for operating it in many places. It is accessibility for easy operation. It is also known and used globally, making it very popular. It supports Wi-Fi which is quite handy for such a small device. There are cons on the server side of Mobile Devices. There is not much processing power to use. It has less security than other devices. It has a battery life which can be inconvenient. The storage is small so youwould have to manage it wisely. |
| **Client Side** | Apple hardware is usually more expensive, but it does offer a good experience using it. It takes time to use, and updates can pop up. Safari and Firefox are available to use and test with. Must know how to use Xcode and frameworks like React native. Must be familiar with JavaScript for web-based browsers. | Relatively cheap as it is open sourced and developments tool are free and directly support frameworks. It takes time to understand and use it for testing. Requires high proficiency to use, frameworks like Flutter should be familiar with, understand their common programming languages, and Linux has specific security practices to learn | There are a lot of licensing fees for tools like Visual Studios and MS SQL Server. Also, for frameworks like Flutter. High end hardware is usually needed. Testing can be time intensive for Windows and Browsers. It requires proficiency in C# and .NET. Also, be familiar with frameworks like Flutter. We should be familiar with JavaScript for web-based browsers. | It is difficult and costly to maintain both IOS and Android at the same time. Must pay for licensing and different tools. Will likely have to get plugins and such to support what you need like Flutter. Maintenance takes time from updates to patches. It takes time to learn as mobile devices are each different in their respective ways. Must be familiar with Swift and Xcode for IOS and Kotlin and Java for Android. Must also be familiar with Flutter and React Native. |
| **Development Tools** | Programming languages commonly used on Mac are Swift, Objective C, Java, JavaScript, and Python. IDE and Dev Tools commonly used on Mac are Xcode, VS code Visual Studios, Eclipse, and Homebrew. Xcode and VS Code are free. There is usually no license fee. | Programming languages commonly used on Linux are C, C++, Java, JavaScript, Rust, and Python. IDE and Dev Tools commonly used on Linux are PyCharm, Git, Eclipse, and GNOME builder. Most of them have zero license fees. | Programming languages commonly used in Windows are C, C++, Java, JavaScript, VB.NET, and Python. IDE and Dev Tools commonly used in Windows are PyCharm, Git/Git Hub, Visual Studios, Eclipse, and Unity. The common versions are usually free but if you want to use them for work you would have to pay. | Programming languages commonly used on Mobile Devices are Swift, Objective-C, and JavaScript  Android- JavaScript, Kotlin, and Java.  IDE and Dev Tools commonly used on Mobile Devices are Xcode for IOS and Android Studio for Android. Others are Visual Studios, Unity, and React Native CLI. Android Studio and Xcode is free. |

## 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; It supports web-based software applications, able to run on anything with a browser. For example, it can run browsers from, Android, macOS, iOS, and Windows. It can also run Apache, Java, and Python. It is open-sourced, stable, and a lot cheaper than other choices.
2. **Operating Systems Architectures**: The Linux operating platform architecture is layered. Hardware in the center then Kernel, Shell, and Application. Hardware has the actual physical components like RAM and CPU. The Kernel is the most important part; manages the device, resources, memory, and handles most of the complicated parts. The Shell is the interface and the one who accepts commands and oversees components like the mouse and keyboard. The Application is what the user would run programs on like web browsers.
3. **Storage Management**: Linux uses LVM (Logical Volume Manager) for storage management. Its backups your data, optimizes storage for specific jobs, simplified partitioning for better management of space, and able to reduce or enlarge volumes when needed. It also offers RAID and snapshots which can help Draw It or Lose It with storing and protecting drawings and user information.
4. **Memory Management**: Linux uses different memory techniques. Virtual Memory is one; each process gets its own space, so memory is safely stored. A Page Cache is when you frequently visit data and that gets recognized by the system to accommodate that; speeding things up. Slab Allocator is assigning memory chunks to where it is needed instead of fragmenting it; to speed things up.
5. **Distributed Systems and Networks**: To connect everything needs to be operated on multiple platforms you need to rely on different things. Web Sockets for multiplayer, Reliable Internet Protocols to keep data transfers stable, and Content Delivery Networks to speed up transfer of images and such. Considering the dependencies between components within distributed systems and networks. Frontend and Backend rely on each other for the game. Backend and Database to keep up with storage that is needed for the game. When facing different issues, it must be dealt with as much as possible. Network outages may require switching to offline and trying to reconnect. Server Crashes can be dealt with is to have backups and a way to recover.
6. **Security**: Security is important as vital information cannot be let out, there are ways to prevent such occurrences. Multi-Factor Identification helps to make sure you are really you. Data needs to be encrypted when resting, full-disk encryption, when transferring, only using HTTPS and TLS, and for real time action use encrypted Web Sockets. User login must be secure on multi platforms, use the same levels of security on all platforms. It should all be regularly maintained to prevent issues arising. Linux has built in security like SELinux, access controls to limit privileges, Firewall, prevents unauthorized traffic, and User Privilege Model, which limits permission to a select few. Linux is also open-sourced so the community can all join to fix issues.