

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/21/22 | Nicholas Glover | Filled out the Executive Summary, Design Constraints, and Domain Model section |
| 1.5 | 06/03/22 | Nicholas Glover | Filled out the Evaluation section |
| 2.0 | 06/17/22 | Nicholas Glover | Filled out the Recommendations section |

**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)

The Gaming Room wants us to design an environment for their game Draw It or Lose It that will support the multiple teams and rounds of the game. In order for the environment to be effective, it will need to support multiple teams with multiple players and maintain the information in a well-organized fashion. It will ensure that teams and players will have unique names so that confusion is reduced.

## [Design Constraints](#_2et92p0)

* Player, team, and game names must be unique
* Multiple games, teams, and players must be supported
* Only one instance of the service should be created

## [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 driving class of the project is ProgramDriver that uses the class SingletonTester to test out the Singleton structure of the GameService class. The GameService class is a Singleton structured class that can contain multiple Game objects. The Game objects can hold multiple Team objects, which in turn can hold multiple Player objects. The Game, Team, and Player objects all inherit from the Entity object. The Entity object stores the necessary id and name information for each object. This UML Diagram demonstrates the object-oriented programming principles of creating concise objects and being purposeful with inheritance. The diagram demonstrates creating concise objects by including only the required variables and methods in each object and all other info is handled by other objects. It demonstrates being purposeful with inheritance since the Game, Team, and Player objects all inherit from the Entity object, they all have the methods and variable for IDs and names without having to fill them in with copied code.

**"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** | Mac OS isn’t a very popular choice to host a web-based software application unless the application is being developed for Mac OS or iOS. It only runs on Apple hardware which increases the cost of using it and developing for it considerably. | Linux is a very popular platform for hosting a web-based software application because it is relatively inexpensive since it doesn’t have licensing fees and very stable, as it’s been around for a while and users have implemented many patches and updates. However, it takes longer to find solutions for bugs and requires more technical knowledge for use. | Windows is a popular choice for hosting a web-based software application since it has a very intuitive GUI out-of-the-box and it has access to a wide variety of Windows-specific services. However, it is much more expensive due to requiring licensing fees and it is a little less efficient than Linux. | Mobile devices really don’t have the hardware capabilities for hosting any usable web-based software applications. However, they can certainly be used for developing client-side software. |
| **Client Side** | Costs to develop with Mac would be relatively high because the OS isn’t open source and the expertise level needed would be moderate. The amount of time development would take depends on the expertise level of the developer. A developer who is more skilled with Mac would be able to finish faster. | Costs to develop with Linux would be relatively low since the OS is open source. The amount of expertise needed would be relatively high since it isn’t as intuitive as Windows and can be difficult to use at times. This would also lend to a longer than usual development time. | Costs to develop in Windows would be relatively high due to the OS being somewhat expensive. The expertise needed to develop in Windows would be moderate because the OS is very popular and commonly used. The development time would be moderate, although it could change based on the developer’s expertise level. | Cost and expertise would not be very large hurdles when developing with mobile devices. The big issues would be time since the variety of devices and operating systems would slow things down. |
| **Development Tools** | Mac has access to a rather limited set of programming languages. Really the only ones available are the ones that specifically pertain to Mac OS devices like Swift and Objective-C and the languages that utilize a virtual machine, like Java and Python. | Linux supports development using almost any language either natively or with SDKs that can be downloaded. The only languages that it isn’t able to develop with are the Mac OS languages and the languages that are Windows-specific. It runs most programs and scripts far faster and more efficiently than Windows or Mac. | Windows can handle development in almost any language and offers access to Windows-specific languages like C#, Visual Basic, the .NET Framework, and ASP.NET. The Windows-specific languages are available through the Visual Studio IDE. | There are a wide variety of languages used in development for mobile devices. For the development for iPhones and other iOS devices the main languages are Objective-C and Swift which will require the use of Apple hardware. For Android devices, the main languages are Java and Kotlin. The most common IDE used for Android app development is Android Studio. |

## 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**: I would suggest that the server-side of the application, Draw It or Lose It, should be run on a Linux server. That’s because the application does not need any of the Windows-specific features that a Windows server would offer and it would run much more efficiently as well. In addition, using a Linux server would be much less expensive than dealing with the licensing fees that come with using Windows.
2. **Operating Systems Architectures**: Using a Linux server would be very beneficial for the game application, because it would allow changes to be made without requiring a reboot. Since a core aspect of the game is the image library, that would allow updates and patches to be made to the library and library handling code with a minimal impact on the players.
3. **Storage Management**: I recommend using a cloud server to store the code that doesn’t have to be on the local client machine and the full image library. That way, the physical size is much more manageable for players and the image library could be updated easier. The library would be in one place and only a handful of copies would be made for each game on the local machine, meaning the image library on the cloud server could be changed or added to with little to no effect on the current players.
4. **Memory Management**: For memory management I suggest copying a 1 or 2 games worth of images from the cloud server to the local machine’s RAM. That way, the games run much smoother since the read/write times of RAM are much higher than that of a cloud server or HDD. It also means that the image library could be updated or changed without it affecting players because the local machine wouldn’t need to be in contact with the library at every moment in the game. Linux also offers many different utilities to help with memory and resource management.
5. **Distributed Systems and Networks**: Draw It or Lose It is already coded very well to communicate between multiple platforms because it’s written in Java. That means it can run on Windows, Linux, and macOS since the Java language runs in a Java Virtual Machine and isn’t reliant on the local machine’s operating system. Using the server-client design pattern also helps by setting the program up for handling multiple users at the same time.
6. **Security**: As a whole, Linux is much more secure than Windows. This is because Linux employs more ways to secure data and offers something called SELinux (Security Enhanced Linux) which helps limit the ways a service can access a server. That way user information can be protected for any and every game of Draw It or Lose It. It would only be able to be accessed if the service requesting it had the correct permissions.