

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 03/21/2022 | Lewis Quick | Updated Executive Summary, Design Constraints, Domain Modal, Evaluation, and Recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room would like to expand their game, Draw it or Lose It, exclusively available for Android and make it web-based to operate on multiple platforms. However, the staff at The Gaming Room are unfamiliar with how to set up the environment for a web-based application and need help to streamline the development.

## [Design Constraints](#_2et92p0)

The issue is that The Gaming Room's game, Draw it or Lose It, is developed for the Android OS and needs to be converted to be developed for a web-based distributed environment. The software requirements for the game application are it must have one or more teams, each team will contain multiple players, game and team names must be unique, and only one instance of the game can exist in memory at any given time. Implementing the singleton design pattern to the class structure will ensure that the game and team names are unique and that only one instance of the game can exist in memory at any given time.

## [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 Gaming Room UML Diagram represents the, Draw It or Lose It, game implementing the singleton design pattern. The singleton design pattern ensures there is only one instance of a Game object in memory at any given moment. The Gaming Room UML Diagram shows the bidirectional association between the GameService and Game, Game and Team, and Team and Name. For each bidirectional association, there can be zero to many instances of its' respective associated object. The straight line with a closed arrowhead pointing from the Game, Team, and Player subclasses to the Entity superclass represents inheritance. The Entity class has attributes and methods to get an object's unique ID and name utilized by the Game, Team, and Player classes representing encapsulation. The straight line with a closed black arrowhead from the ProgramDriver to a SingletonTester shows a unidirectional association. The ProgramDriver knows about the SingletonTester but not the other way around.

**"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** | The macOS server makes it simple to configure other Mac and iOS devices. The macOS is relatively stable in comparison to other platforms. The macOS can be expensive to obtain licenses. | The Linux OS is the most cost-effective environment. It has a robust, reliable, and stable environment to host web-based software applications. | The Windows OS Offers a user-friendly GUI to simplify hosting a web-based application. Windows OS is known to have system crashes and bugs. | Mobile operating systems like iOS and Android are the least desirable for hosting a web-based application. However, a mobile OS can significantly increase the functionality of mobile devices to interact with its environment. |
| **Client Side** | The macOS requires the client to have a MAC device and can be costly. The macOS requires a reasonable amount of time and expertise in comparison to other platforms. | Most of the software for the Linux OS is open-source, which makes it the most cost-effective option. The Linux OS Requires a significant amount of expertise and knowledge of the command terminal. | The Windows OS has a plethora of software for software development, such as IDEs. The Windows OS requires minimum time and effort. | Mobile devices have significantly different hardware than the other OS devices. The biggest difference is screen sizes and the limited amount of space available to display web-based applications. Mobile devices will require extra time to format the application. |
| **Development Tools** | The macOS requires software for iOS and Macs written with either Swift or Objective-C. If using Swift, the macOS requires that you use their IDE called Xcode. | The Linux OS supports almost all programming languages such as C++, Java, Python, HTML, CSS, and more. The Linux OS also supports many of the popular IDEs like NetBeans, Eclipse, and Komodo. | The Windows OS is probably the most widely used operating system for development and supports almost every programming language and IDE. The most dominant language for Windows is C++, and when combined with Visual Studio, it is a force to reckon with. | Visual Studio is the best IDE for mobile cross-platform development using C# and XAML. However, mobile development typically requires that iOS and Android applications get written in their native languages. Java is Androids native language, and Objective-C and Swift are the iOS native language. |

## 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**: The Gaming Room wants to convert their Draw It or Lose It game from Android to a web-based application that serves multiple platforms. Based on staff at The Gaming Room and their level of expertise, I suggest utilizing the Windows operating system. The Windows operating system has robust tools and software to expedite and simplify the shift from Android to a web-based application.
2. **Operating Systems Architectures**: The Windows OS has a layered Operating System Architecture consisting of User and Kernal spaces. The Kernal space is layered and consists of I/O management, Message Interpreter, Memory Management, and Processor Scheduling. The User space is layered and consists of User applications, Environment subsystems, Service processes, and Subsystem DLLs. The Windows OS Architecture allows user applications to display a GUI and access system resources.
3. **Storage Management**: The Windows OS maps files onto physical media such as a hard drive or solid-state drive. The Windows OS utilized a Disk Management utility that enables a user to perform advanced storage tasks.
4. **Memory Management**: The Windows operating system handles the mapping of addresses during memory allocation. The Windows OS enables a Memory Manager that maps virtual addresses to physical addresses during runtime. The Windows Memory Manager reduces the amount of memory while letting the operating system maintain more physical memory.
5. **Distributed Systems and Networks**: The database is an essential component for the Distributed Systems and Networks of the Windows OS. Access to a shared database and network will improve speed, functionality, and reliability. The Windows Distributed System and Network are composed of individual nodes, and a client prompts operations.
6. **Security**: The Windows OS has its' own security applications that scan for viruses, malware, and security threats. The operating system's security is vital for hosting a web-based application since many users will be accessing the server at once, making it vulnerable to malicious attacks.