

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.0 | 06/17/2023 | Felix A. Carela | None. |

**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 their game Draw It or Lose It developed to work on a web browser, this web-based version of the game will be able to be played on multiple platforms. The game is currently playable on Android only. In this game teams compete to guess what the image being rendered by the computer is. This game will have the ability to have one or more teams involved, and each team will have multiple players assigned to it. The 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. Creating one instance of each game will be possible by using the singleton design pattern.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

This game will be a web-based game that runs on multiple platforms, designing a web-based game that functions correctly on multiple operating systems will require additional knowledge on those platforms. This game running on multiple platforms will also require additional knowledge to develop this for different web browsers since certain platforms mainly use their own web browser. This means that different programming languages for different web browsers and platforms will be interacting with the same back-end structure, which could require additional optimization and testing.

## [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 ProgramDriver class contains main and therefore drives the program while using the SingletonTester. GameService is an interface class that references Game, Game references Team, and Team references Players. These classes reference each other by creating instances, however, since there is a 0…\* amongst these four classes then there may be 0 or more instances at one time. Inheritance is used within the UML diagram since Game, Team, and Player inherit from Entity which is done to prevent writing the same variables and methods multiple times. Encapsulation is also used here since Entity has a private constructer which prevents other classes from making additional instances of Entity.

**"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 OSX would be able to host a web-based software application. Mac has a lot of capabilities and a lot of support since the Mac user based has steadily increased in the past decades. Mac does have a limited variety of hardware that is almost all proprietary, this limits the server-side hardware options. | Linux is an open-source OS with a lot of capabilities to modify software in order for it to work. It does have an active community for support, and this OS can be run in almost any PC. Its disadvantage is that out of Windows and Mac it is the least used. This means that there is less user synergy, meaning less support, bug fixes, and compatibility. | Windows is the most common OS, therefore, offering the most support, biggest community, and compatibility. Most of Window’s hardware is also not proprietary, meaning that there is a wide variety of hardware to choose from. | Running a server on a mobile device is not recommended, since they will usually not have enough computing power to keep up with demand. Other devices would be much more advantageous. |
| **Client Side** | Software development on the client side for Mac will depend on the expertise on the specific programming languages required for this OS. An added cost for Mac is that their products go for a premium of what Windows goes for. Specifically, since Mac has proprietary hardware, the user will have to buy a Mac product to use the Mac version of this game. | Software development on Linux can be expensive since it is not a common OS. Since it is not common, then it is also not as cost effective as an OS that is common. This is because you would be spending around the same or more money to develop the client side while at the same time having less people using the software. | Developing a client for windows would probably be the cheapest since it is the most common. Even most Mac users know how to use a Windows OS. Windows would probably be the most cost effective since most users are on Windows. | Mobile devices are widely use and software development for them are cost effective considering the amount of users. The experience of developing for mobile devices should not be any more difficult than other platforms, since mobile devices have smaller screens and therefore less details to work around. Additionally, since the game is already developed for Android, IOS is the only other primary market that the game will have to be developed for. IOS comes in limited devices therefore, making development easier. |
| **Development Tools** | For Mac, the primary language is Objective -C, with Swift also being of use. However, other programming languages and IDEs can be used to develop apps for a Mac. | Linux does not have many limitations to the programming languages that can be used. But IDE support can be an issue since Linux is not a widely used OS. | Almost any programming language can be used in Windows, as well as almost any IDE. Windows is widely used and has been around for a long time, giving it an advantage to the number of programs available to it and the amount of support available as well. There are specific programs that cannot be used for Windows such as objective-C and Swift. | Java would be used to develop Android apps; however, the Android app was already developed. Leaving the need to use only Swift to develop the IOS app. |

## 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 the client’s need to allow The Gaming Room to expand Draw It or Lose It to other computing environments; I recommend Linux. Linux is a popular server platform that offers both compatibility and scalability. It also benefits from a strong open-source community, which provides support with development and expansion. Ubuntu Server is a robust system that can be deployed on a variety of hardware architectures and supports a wide variety of applications.

**2. Operating Systems Architectures**: Linux is a monolithic kernel since it contains all operating system functions in the same address space. The architecture of Linux allows it to run a large number of processors, while also simplifying the management and coordination of various system components. Linux has two main components, which are kernel space and user space. Kernel space is a privilege area of the operating system, while user space is where user applications and processes run.

**3. Storage Management**: A good storage management system will be using a hard disk drive. The size of the game file including the 200 pictures is not big and can be all loaded into main memory. However, since main memory is volatile then it will have to be initially stored in permanent storage such as the hard disk drive mentioned. Since the server will presumably run at all times not including schedule maintenance, then the storage will not play a big role on running the server side of the game. This is because its main memory is able to get the files from storage memory, and then just keep the entire game file in main memory for faster use.

4. **Memory Management**: Linux employs various memory management techniques such as virtual memory, memory allocation, and demand paging. Linux allows each process to have its own virtual address space. This allows processes to have access to larger address space than the physical memory available, due to the combination of RAM and disk storage. Linux also uses various algorithms for memory allocation, such as buddy system and slab allocation. These provide different ways for memory allocation and deallocation. Lastly demand paging is a technique that loads pages from storage memory to main memory only when they are needed by a process.

5. **Distributed Systems and Networks**: This game has one server with multiple clients using different web browsers. The architecture needed must connect multiple devices to the server using a form of networked system. Technologies such as TCP/IP or UDP/IP protocols would work best to achieve communication to the server and vice versa. These protocols would work best in a web-based game since a web browser will be used to process requests, almost the same way web browser page requests are handled. The dependencies within the distributed systems and networks will be potential outages and network connectivity.

6. **Security**: A multi-layered security approach should be used in order to protect information on both the client and server side. Data encryption is a necessary way to secure information between systems. A firewall for the Linux server will also be necessary in order to allow only necessary incoming and outgoing network connections. Password policies is another effective way to enhance security, since this will prevent the users from using easy to guess or crack passwords. After securing a system there may be additional vulnerabilities created with new updates and technologies, so regular security audits should be scheduled. It is impossible to completely secure a system; however, it is possible to make the costs for attackers to be too high for them to attempt breaching the system.