

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | <05/23/24> | Blake Hardrick | Initial version. |
| 1.1 | <06/23/24> | Blake Hardrick | Added recommendations |

**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 to design a game called “Draw It or Lose It.” The players of the game will be divided into teams. Each team gets a turn that consists of an image being rendered over 30 seconds. They must guess what that image is before the time runs out. If the time does run out, the other teams get 15 seconds to guess the image. The game has a total of 4 rounds before being finished.

## Requirements

There must be support for several players. They must be grouped into teams. The application must be able to keep track of time. There needs to be images slowly rendered.

## [Design Constraints](#_2et92p0)

This application must be designed to work on the most popular web browsers so it can have the most amount of players. The code written must be cross platform, and able to facilitate the client-server model. Also, it must be able to render images and keep track of 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 ProgramDriver is the main class containing the main function. It uses the SingletonTester to test if the singleton works and if it was properly written. The Entity class is the base class for Game, Team, and Player, because they all have a unique identifier and name. Each game is managed by the singleton GameService. Each game has several teams within it. Each team has several players on it.

"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 is a good platform for hosting the server side code because it’s high quality hardware. It is also built upon the Unix platform so software written for the platform is stable. However, Mac’s are relatively expensive compared to the other alternatives, especially if the server needed to be scaled to support more players. | Linux is also built upon the Unix platform. This makes it a reliable option. Depending on the hardware that Linux runs upon, it could be a cost effective and reliable server side option. Investing in high-quality hardware to run the server on Linux is a scalable and affordable option. | Windows is a good option because of its popularity and friendly user interface. The platform has a dedicated Windows Server. This comes with some licensing fees, however, and it may not be as cost effective as something like Linux. | Mobile devices are not designed to run server side applications. The devices are small and heavily optimized to run client side code for small screens. |
| **Client Side** | Due to the proprietary nature of Mac and developing applications for the platform, it is an extra expense to support because you need to have a Mac device to publish the app with. You need to test the app using Safari to make sure that browser works with the server application. | Linux is an open source OS built upon the Unix platform. The cost would be free to obtain the OS. Linux doesn’t come with a Web Browser, so testing with the most popular web browsers such as Firefox, and Google Chrome would be good for this platform. | Windows would be the easiest computer platform to support due to its popularity. Ensuring the server side code effectively communicates with the built-in web browser Microsoft Edge is a must. | IOS and Android should be supported because many people play games on their phone. It would be extra expense to publish a web app to the respective app stores, but supporting the Safari browser on iOS and Google Chrome browser on android devices shouldn’t take much more time because those already have to be supported for the other platforms. |
| **Development Tools** | For Mac Eclipse can be used to develop the server-side code and Safari to test the client. | For Linux, Eclipse can also be used, and any web browser to test. | Eclipse can be used for server-side and Microsoft Edge can be used to test the client. | For mobile devices, use the built-in web-browsers to test the client. |

## 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**: After considering the various operating platforms available to run the server on, I would suggest using Linux. It has a variety of development tools available to it, and the OS doesn’t have too much overhead for running a server. Because you would want to expand the amount of supported users in the future, running servers on Linux is a good option because it is scalable.
2. **Operating Systems Architectures**: Linux supports many architectures. As far as hardware, many different CPU’s can run the OS. X64 is one of the most popular for 64-bit systems. Another option is ARM.
3. **Storage Management**: The only things that would need to be stored on the server are the server application and the images to send to the clients for display. The images can be stored on a hard drive and loaded when clients connect to play the game. 2 GB of storage capacity for the application plus what the OS needs would be plenty of storage capacity, because the storage will not scale with how many players.
4. **Memory Management**: The server will not need a lot of memory per player, but it will need more memory if there are many games going on at once. Different images may be loaded for different games going on at the same time, so the memory should scale. RAM usually comes in 8 GB increments, so 8 GB is a great starting point, and once about 75% of that memory is being used, more RAM should be added. For every game that is created, there should be enough space allocated for one image. That image should be loaded into the space allocated and sent to the client.
5. **Distributed Systems and Networks**: There has to be a client app created for each platform that wants to connect to the server. It would be a web based application so they can connect to the server and play through their web browser. This would use an HTTP connection. This would maintain a good connection unless there was a server outage. If we wanted to avoid server outages, we could have multiple servers to possibly connect to. Otherwise we could have a backup generator to ensure the server doesn’t lose power.
6. **Security**: The Linux OS provides many security features. File permissions only allow the users with proper credentials to access files. The Linux Kernel is consistently updated to avoid attacks. Linux also utilizes a secure shell (SSH) for remote access to the server. For security over the network, using the HTTPS protocol will protect the client’s information.