

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 | 10/14/22 | Tiffany Montero | Updated recommendations for client. |

**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 client wants to develop a web game based on their Android app. Teams compete to guess what is being rendered from stock images. The game is four rounds and if the team does not guess correctly within the time frame, the remaining teams will have 15 seconds to solve the puzzle.

## [Design Constraints](#_2et92p0)

Only one instance of each game can exist, which will require a singleton pattern. Game names and team names must be unique. Multiple teams and players have to sign up to be able to play.

## [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 uses the SingletonTester to ensure only one instance of the game exists. The Entity class is the parent class to Game, Team, and Player which creates a relationship between them. All 3 classes will also inherit Entity’s characteristics. Encapsulation is also shown with the private attributes and methods that are included.

**"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** | MacOS is very flexible and can be configured to meet almost any need. However, it is not as commonly used as other platforms. | Linux is cost efficient. The biggest drawback would be that less users are very familiar with this platform, so it would require finding someone who is confident with Linux. | Windows is a great choice for hosting a web-based software application. It supports the most programs and languages and is most commonly used. | Since specifications vary between each mobile device, it would be more challenging to build a game that works properly on all devices. |
| **Client Side** | More expertise is required for MacOS. It is also more tedious to find MacOS specific instructions if needed. It is very efficient once you are familiar with the process. | Since Linux is not as common, you would have to find someone who is well versed, which may take longer than it would for other platforms. However, it is the most cost efficient. | Windows is not open source, so initial set up may take longer. However, it is the most common platform, so it would be much easier to find someone who is experienced. | Experience is most likely not going to be much of a challenge, since mobile applications are very common. The biggest challenge I believe would be developing the game for varying operating systems in mobile devices. |
| **Development Tools** | Swift is the most popular to use with MacOS. You can also use Visual Studio Code, eclipse, etc. It also supports HTML, CSS, JavaScript, Python and Java. | Seamonkey is a very user friendly tool. You will most likely need knowledge in HTML, CSS, JavaScript, Python, and Java. | Eclipse and Visual Studio Code are both very common to use with Windows. Languages most likely to be used would be HTML, CSS, Java, and JavaScript. | To develop an app on iOS, you would most likely use Swift, which requires a device running MacOS. For all others you can use Visual Studio Code or any other IDE. Languages used would include HTML, CSS, JavaScript, among possible others. |

## 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 research and consideration, I recommend for The Gaming Room to start with Windows. The operating system is the most widely well-known and used, which would help lower cost. There are also many IDE’s available to use on Windows.
2. **Operating Systems Architectures**: Windows architecture is very user friendly and includes a GUI (graphical user interface). It has a variety of IDE’s, programming languages, and packages available for developers to use, which helps customize to best suit your needs.
3. **Storage Management**: Windows 10 now comes with a feature called Storage Sense. You can choose how often it should automatically clear any temporary files, downloads, pictures, etc. You can also choose to disable it if you would prefer to manually clear unnecessary files. This can help you maximize your storage space and avoid from the cluttering that tends to happen over time and regular use.
4. **Memory Management**: 32-bit Windows can accommodate up to four gigabytes of memory in its own virtual address space. This will help free RAM (random access memory) to maintain processing power.
5. **Distributed Systems and Networks**: In this case we would want to use a client-server system. The server would have all of the information regarding the game, its processes, and any images that will be rendered. The client would be the user, who would send commands for the server to respond. Multiple clients can connect to one server. As the game grows, it’s crucial to anticipate growth before the server slows down due to too many users. This would present itself with lagging on the user’s side.
6. **Security**: Windows comes with built in security and antivirus tool called Windows Defender. Although it would be highly recommended to get third party security tools such as a stronger antivirus and maybe even a VPN. Most tools out there are compatible with Windows so it would be possible to find something to meet our needs to prevent malware.