

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 | 03/17/23 | Kenneth Jarvis | New software requirements |

**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 has hired CTS to develop their web-based game "Win It or Lose It." They would like to service multiple platforms as it only is available on Android at the moment. This platform is similar to the 80's TV game "Win, Lose, or Draw." Teams compete to guess on what is being drawn. Instead of contestant drawing, the app will use images from a library as clues. There are four rounds, one minute each. Drawings are rendered steadily for 30 seconds. If the team can't figure out the puzzle, the other teams will have a guess with a 15-second time limit.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* 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.

## [Design Constraints](#_2et92p0)

The game needs to run on multiple platforms, while still compatible with Android. Rewriting code will have to happen on some devices like Apple. Some platforms may be able to port over code from Android. All devices will probably want to play together in a cross-platform format.

## [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 Entity is the superclass, while Game, Team, and Player are its sub-class. Sub-classes inherit information like "id" and "name" from the superclass. The subclasses reference each other with aggregation (Games has a team and GameService has Games. Team and Player both have types). GameService [references] Games [references] Team [references] Player. The SingletonTester class has a relationship with the ProgramDriver Class.

**"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 will have no problem hosting a web-based software application, it might be the most stable out of all platforms. The cost of buying Mac hardware would be higher than on the PC side. Mac is a closed-ended software company, it will be more reliable but also cost more to develop the software through licensing fees. | Linux is great at hosting software. It will run perfectly if set up correctly. But there can be hardware compatibility issues and would take the most amount of time to set up overall. Although the cost for licensing would be free. | Windows is the most popular platform and is moderately reliable. Windows has been known for security issues and server crashes. But if enough hardware is available, it will work fine. The cost is similar to Mac, but the time to setup would be minimal. | Mobile Devices would not be a good option for running a server-based web application. There is not enough CPU processing power or ram. Mobile devices are intended for one client, not hosting many clients. |
| **Client Side** | Expertise would be needed as not many use Mac compared to Windows. Time would be moderate, there is is not too much adjusting needed with Mac," it just works." | One slight problem could take days to resolve. There would be a need for expertise as there is usually minimal documentation for Linux. | Minimal expertise would be needed as Windows is the most popular and there is plenty of documentation. | Android is already set up, but Apple might be able to be ported. If not, expertise would be needed. And it will take more time. |
| **Development Tools** | Swift would be the most popular language to use. And Atom is a good IDE and free. Eclipse is free and work for Mac, Windows and Linux.  Using the same IDE can impact the development team positively, it would be easier to transition from one platform to another inside the same IDE. Just switch projects. | Java, Python, and PHP are good languages to use with Linux. Eclipse, and PyCharm are good IDEs and free | Microsoft makes Visual Studio for a IDE, but others are available depending on the language you want to use, most are free. Java, Python, HTML, C++, and JavaScript are a few that can be used. | Swift for iPhones can be used, so Atom for the IDE, it is free. The Android app is already developed, but Android Studio is good for development. Java, BASIC, and Kotlin are popular with Android. |

## 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 recommend Windows as their Operating Platform. There are many IDEs available, and it requires the least amount of expertise. There is plenty of documentation out there since it is the most popular OS. Cost for experience would be lower, compared to others.
2. **Operating Systems Architectures**: The Windows Architecture allows direct access to the GPU and RAM, it is already the most popular gaming platform because of its features. It can handle the heavy payload of a game while keeping the system stable. It is also capable of running other servers such as email, web server, and more.
3. **Storage Management**: Storage sense is a great tool for storage management. You can manage files on your hard drive, and storage sense can automatically get rid of items you don't need. Temporary file, Recycle Bin, and more help you have less cleanup. Having a backup of all data on an off site is also crucial, and redundancy will also help with data loss.
4. **Memory Management**: The game's image files must be sorted in a database and distributed randomly. Using memory will allow the images to be pulled faster than if in storage. So, keeping images in memory for others to use will help with the game's speed.
5. **Distributed Systems and Networks**: Using a cross-platform IDE can come in handy. Xamarin works with iOS, OS X, Android, and Windows. It uses C# so it will be more efficient to use this platform. HTML5 might be a better option as it is web-based and can work in most browsers in most OSs. A strong connection to the internet will be needed to pull from the image database. A newer computer will be helpful to handle newer graphics. A battery backup for the servers will be needed for intermitted power loss.
6. **Security**: Windows has a built-in firewall(software) and antivirus protection. Security will still need to be set to high in order to require a password for any changes to be made. A second firewall (hardware) will be needed for extra security. The second firewall will shut down all unnecessary ports for better protection. The antivirus will need to be updated constantly as new threats develop every day. Real-time protection will require to be enabled to display any compromises as soon as it happens. Security keys will need to be distributed to authorized personnel only. Using security keys will allow two-form authorization and make a compromised password alone useless.