

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 05/17/2023 | Kiersten Grove | Updated recommendations |
| 1.1 | 06/01/2023 | Kiersten Grove | Second draft, updated recommendations |
| 1.2 | 06/15/2023 | Kiersten Grove | Third draft, updated final 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 wishes to develop a web-based game that has the capability to run on multiple platforms. The name of the game is “Draw it or Lose it” which is currently only available on android. The purpose of the game is like Pictionary, in that it consists of multiple teams trying to guess what is being drawn with four rounds at a minute each. Drawings are rendered at a steady rate and are fully complete in 30 seconds; if the team does not guess the picture within the one-minute time limit the remaining teams each get one guess to solve the puzzle with a 15 second time limit.

A singleton creation pattern has now been adopted for object creation. This singleton pattern prevents multiple instances of the game. Along with the singleton pattern an iterator pattern was also adopted which will prevent conflicting teams and team members.

## Requirements

The client has requested the below requirements:

* The game needs to have the ability to have one or more teams involved.
* Each team shall have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use.
* 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)

Following the requirements given we can conclude that the project may be a bit more costly as they want the game to run on multiple platforms. This will mean that multiple development teams may become necessary, one team to write the code of the game for their respective operating platform. Android is already available however this means the game will need to also become compatible with iOS phones, Mac computers, Windows, and possibly even Linux.

Any existing APIs that are currently serving the Android platform will need to be reviewed or extended for mobile usage.

## [Domain Model](#_8h2ehzxfam4o)

The Game, Team, and Player classes each inherit from the entity superclass allowing the attributes and operations contained within entity to be written once and used within each of the child classes inheriting from it. All classes present, GameService, Game, Team, Player, and Entity are all associated with each other. These classes use a zero to many association which means they can use as many instances of each other as necessary or none at all. The ProgramDriver class drives the package and inherits the SingletonTester class while doing so.

**"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.

The table below discusses the strengths and weaknesses of each OS platform for either hosting a server or acting as the client. Server and client choices are not linked however, so if Linux was chosen for the server then the client may still prefer Windows or Mac on the client side.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Has flexible terminal commands to use in configuring the server, the accessibility, and make changes.  OS X server is available for Mac however the client would have to purchase their own hardware in order otherwise finding hosts will be difficult and expensive. | Has flexible terminal commands similar to Mac however, is less costly than Mac.  The most popular OS as it is open source. Maintenance and license costs tend to be less expensive than closed OSes. | Has more software available than Mac or Linux due to it’s higher user count and compatibility with third parties.  License costs tend to be per user and tend to be very high compared to Linux. Hosting platforms may be more limited as well, compared to Linux. | Specifications are per device and cellular provider. Immobile servers are easier to keep track of.  Hardware is more limited than standard computers. The costs are unknown. |
| **Client Side** | Moderate expertise is needed, Mac is known as being user friendly, however, it is the most expensive option and does not have open compatibility with third party programs.  Development for Macs requires a Mac device that is running the latest version of XCode. Code is also in Objective-C or SWIFT which are lesser known languages. | Maximum expertise and time required as the user has maximum control over the interface and functionality.  Development is straight forward. Java, C/C++, or Python could easily be the language of choice. Multi-user support is also available. | Minimum expertise and time required. Windows is the most affordable option for users. Has maximum compatibility with other programs and third-party operations.  Windows is a native multi-suer and is based off of C# or .NET which are both very common. | Provides flexibility to clients, updates can happen anywhere at any time and in minimal time. Ease of use depends on the mobile device and the cellular service provider.  Mobile devices are not meant ot be multi-user however designing applications is straight forward. Android SDK is java based. |
| **Development Tools** | Mac has the capability to run all languages however the interface has changed to become compatible with iOS. This is fine when development exclusively on Mac but causes issues when trying to work on a project primarily being built on Windows or Linux because the languages are not fully cross compatible. | Linux also has the capability to run all programming languages including visual studio, HTML, CSS and more. Linux also offers better support for some programming languages and can often be the preferred OS to use in programming. | Windows offers a wide range of language usability. It also is more user friendly since it is the most used OS. | Mobile devices can be used to create apps. Kotlin is most popular among Android and Ios of course for Apple iPhones. Supporting libraries are used in the front end of the applications. |

## 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**: My recommendation would be to start creating the game on the windows platform. It’s the most cost-effective route with many IDE options.

Linux offers good security and operability as well as being the most common server platform. The front end of the program is going to be agnostic and can be written in the preferred language for the varying platforms.

1. **Operating Systems Architectures**: Windows provides the most cross compatible services for each platform requested. Window based applications allow a user to enable Graphical User Interface (GUI) while accessing system resources. These applications also refer to graphics and multimedia messaging, and web services which can be utilized via a user account or server.

A modern backend running containerized services would allow for scalability. A cloud provider would need to be selected before the exact architecture was determined. Using the frontend for rendering allows the server to offload some of the more resource intense parts which reduces monthly data costs.

A serverless cloud computing system is the final recommendation to The Gaming Room. This is the most cost effective option and eliminates any issues that may occur along the lines of cross-platform development. This paired with it’s unwavering affordability makes it the best option to expand Draw It or Lose It across other platforms.

1. **Storage Management**: Windows offers cloud storage with a silent assistant called storage sense. Storage sense communicates with OneDrive to make local files you no longer use online-only files that can be accessed on your OneDrive. You have full control over what applications OneDrive will sift through for you which makes it a great storage companion on top of local disc storage.

A standard SSD or HDD would be plenty to provide the performance needs of the application paired with a caching behavior and client side rendering.

To go along with a cloud server a virtualized storage method would be best. This allows The Gaming Room to adequately allocate funds into memory storage without overpaying or overbuying.

1. **Memory Management**: In creating this game the developer will also need to create a library or database of pictures. Memory allocation allows for easy storage outside of the default picture folder. Separate folders allow for a more secure area to place pictures that will be used in your projects. This also pertains to the IDE in use and how to open the files for the project.

Linux implements the concept of pagecache for data stored in main memory. Linux also uses paging to allow for lower memory usage because things not actively being used will not be loaded into memory.

Android Runtime uses paging and memory-mapping to manage memory. With this any memory an app modifies remains resident the RAM and can’t be paged out.

Virtualized memory performs best with a serverless cloud architecture. Direct access storage would be best as users will be utilizing their random-access memory (RAM) to play the game from whatever device they’re using. Direct access will allow for better use of RAM, regardless of platform, in the virtualized environment.

1. **Distributed Systems and Networks**: In searching for an answer on how we can ensure cross-platform gameplay I found CLion by JetBrains. It’s a C and C++ based IDE that allows a user to develop C and C++ on multiple platforms. A development team could create the game in Windows and then export the files to the other platforms. This helps with dependencies and allows for time saved in development. The Gaming Room will need to ensure their servers are powerful enough to support the cross platform play to avoid potential outages or other connectivity issues as well as a large player base especially with this being a cross platform game, it will be more available to people.

The frontend and backend will communicate via RESTful APIs which allow the client/server communication to be transparent.

Due to the expanse of which Draw It or Lose It may cover it would be best to distribute servers by location to service an area within a certain radius. This will help minimize total server outages and spread out the work load that would occur within a singular server. In the case that one server is down clients will still be able to access other servers and continue to play Draw It or Lose It.

1. **Security**: The best thing to do is ensure the servers are high security servers. Windows OS comes with its own security software, however, to protect clients the security must go beyond the system the game was created on. The Gaming Room could use something like Easy Anti Cheat, a security program that is installed on many online games. It catches unordinary instances such as cheating, hacking, information leaks etc. that may have been initiated by another player or may have come from corrupt install files. This ensures that corrupt data is found and the user cannot deploy the game whether it be for their own safety or the safety of others.

Authentication and authorization are essential for ensuring the security of the end user and the development itself. The user role should only initiate, participate in, and end a game, all other functions are to be left at the use of administrators or higher within the development team. Access to anything outside of general participation in a game will be restricted for users. Password regulations will be implemented and it would be preferable to have users either update their password every 6 months or require a system of two factor authentication upon log in.