

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 | 05/20/21 | Andrew McPherson | Version 1.0 is the starting point. Initial creation of executive summary, design constraints, and domain model explaining the UML diagram. |
| 1.1 | 6/3/2021 | Andrew  McPherson | Version 1.1 adds the evaluation chart, discussing server-side, client-side, and development tools across different Operating Systems. |
| 1.2 | 6/16/2021 | Andrew McPherson | Version 1.2 adds my recommendations for operating system, storage and memory management, as well as distributed systems and how to handle security. |

**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)

Design problems for this app are that there must only be one instance of this game at any time, game names and team names must also be unique and therefore not allowed to be created if one exists already.   
This also means that there needs to be a way for teams to check a name to see if it is available, as there can only be one.

The game must also have four one-minute rounds in it, where during these rounds drawing must be rendered and fully finished after 30 seconds (for the halfway mark of the round).   
Also, if the team fails to complete the puzzle before that one-minute timer expires, the other teams have a chance to solve it with a 15 second timer.

To solve these problems, there must be unique identifiers for the game instance, the team names, and the player names to satisfy the “only one in existence” requirement.

## [Design Constraints](#_2et92p0)

The constraint on this project is simply a software constraint, this app is to be web-based, which requires us to develop the app for that in mind and are unable to develop this for any other platform than web-based. We must also adhere to the choice of software, either chosen by you or imposed on us based on what is available for web-based games. This choice needs to be considered before development of this application.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The class Entity is what stores all the variables to be used for the program, such as id and name. The classes Team, Player, and Game, all inherit from this class. The entity class also contains the methods to get names and ids.  
The Game class related to the Team class, where it may be 0 to more instances, and likewise the Team class is related to the Player class. This is done to add team name, and then add player names to that team name.

The GameService class is what gets the next game, player, and team ID, as well as adds game instances into the Game class. Similarly, as Game relates to Team, GameService also has 0 to more instances into the Game class.

The ProgramDriver class includes main, where the program is run from. This class uses the SingletonTester, and the SingletonTester is what ensures there is only one instance of a specific item (such as a unique game, or team name)

****

## [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, while being only slightly more complex to understand than Windows, generally has a “ease of use” to its system which could make it more user friendly in that regard. | Linux is a more complicated operating system and is less common than Mac or Windows from my perspective. It would take more time to learn but may be a worthwhile endeavor. | Windows has a plethora of more software options and is more commonly used, along side mac. It is also easy to learn and adjust to using. Due to it being more commonly used and your many choices of software, this is a strong contender. | Mobile devices generally do not have the means to host servers on themselves, at least on a large scale of thousands of players, and would be better suited to having the server be immobile. |
| **Client Side** | As said above, understanding the ins and outs of mac could take slightly more expertise than others (Windows). This also translates into time because of the time needed to learn Mac. Cost is like Windows, as while there is open-source software, there is also a lot of licensed closed-source software too. | As said above, Linux is complex, to learn and understand Linux would take considerable expertise and time required to learn it. However, Linux is primarily open-source the costs is considerably lower compared to its counterparts. | As said above, Windows tends to be commonly used and easy to understand, so expertise is minimal. Cost to be considered is (from what I understand) related to licensing costs. Purchasing a license for Windows can be costly, though open-source software is plentiful. | Mobile devices tend to work with many different web browsers, because of this there is some level of flexibility at the cost at being slightly more complex to develop. |
| **Development Tools** | While globally Eclipse IDE can be used on all platforms, for more specifically Mac oriented IDE would be things like Visual Studio, Homebrew, or Xcode. While the primary programing language for Mac is C, C++, or Objective-C | As before, Eclipse is globally used across platforms, other common IDEs are Atom, Sublime, Brackets, as well as Visual Studio Code. As for Programming languages we once again have C or C++, Java, or Python. | Once again, for IDE, we have things like Eclipse or Microsoft Visual studio. Among others would be Code::Blocks or IntelliJ IDEA. For Programming languages, though Java is used, it primarily consists of C, C++, or C#. | For mobile devices for platforms such as Android, we have Android Studio, Xcode, and Visual Studio, and while Eclipse is also compatible with mobile, it does not appear as common. For programming languages we have Python, Swift, and Javascript. While also Java and C++ are used. |

## 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**:

Any operating system may work here and not be too restrictive, aside from perhaps mobile device OS. For this, I would simply recommend whatever is most comfortable to work with and maintain, as anything done for one OS could be done for another. Windows as the chosen operating platform, for example would be my go-to, as it is easy to use and common place due to the familiarity it provides.

(Though at a side note Linux would be another strong option, but due to it being more complex to grasp, a more familiar operating platform like Windows is the safer option.)

1. **Operating Systems Architectures**:

The architecture of windows is a layered design, with two main components, these being “user mode” and “kernel mode”. In User mode, Programs and subsystems are limited in terms of system resources they have access, executing code has no ability to directly access hardware or reference memory. User mode also is what handles I/O requests to the appropriate kernel mode drivers through the I/O manager.

For Kernel mode, kernel in windows is known as a hybrid kernel, which works with the Hardware Abstraction Layer (HAL, a layer between physical hardware of the computer and the rest of the OS). The kernel works with HAL to schedule activities to be preformed by the CPU. Kernel mode stops user mode services and applications from accessing critical areas of the OS that should not have access to.

1. **Storage Management**:

For this, something like physical storage on windows involves physical items such as hard drives (Could be HDD, or SSD. SSD would be something I recommend more, as they are faster) that store all data, files, and other executable information physically and capable of being stored permanently on disks. Windows also has the system utility called Disk Management which enables you to preform advanced storage tasks. Such as initializing new drives, creating partitions on said drive, and even changing the drive letter to something else.

Windows also allows easy access to view and move files stored on a hard drive via file explorer, so all of it is easily accessible for adding to storage or modifying the contents of it.

1. **Memory Management**:

Windows has both physical and virtual memory. Memory is managed in pages, with processes demanding it, as necessary. Pages in system space can only be accessed from kernel mode, whereas user mode processes can only access data that is appropriately marked in user mode. For Draw It or Lose It, memory will be used to allow the game software to run. The images needed for each game will be loaded into memory to be used. Through things like memory paging, where memory is stored or retrieved usually from physical storage such as hard drives for use in main memory. I also believe that information of players and teams for each game in Draw It or Lose It will be added or stored to main memory for use, so memory or RAM will be required for this software (or any) to run.

1. **Distributed Systems and Networks**:

Connectivity through a distributed system would be stable, even if one system were to fail, it would not affect the availability of the service, as other machines on the system would pick up the load and continue to run. A distributed system would be many different machines, either computers, physical servers, or virtual machines that can connect to a network and communicate by passing messages. Different machines could do different things and all work together as one cohesive unit. Using a distributed system would allow various platforms to access this application. Distributed applications are broken into two separate programs, the client software, and the server software. The server software may also be in a cloud environment. And by doing so, the cloud computing can be used instead of servers or hardware to process a distributed applications data or programs.

Distributed programs also allow multiple users to access the apps at once, which would be crucial for this application, not only because of the desire for this application to communicate between various platforms.

1. **Security**:

Security through Windows is no different than other OS, structures such as password requirements or role-based security access would be suitable for the application. You may also layer levels of security, passwords required for entry, and login access for specific roles with specific access. Even if a password were stolen for a game users account, there would not be much anyone could do on such a low level of roles, where each role has a limit to what it can see or do. Security like this would work regardless of platforms of the users.