

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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)3-4

[**Evaluation**](#_2o15spng8stw)4-7

[**Recommendations**](#_m8aleynsvzvc)8-10

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 5/21/22 | Ryan Reames | Updated executive summary, design constraints, and domain model descriptions. |
| 1.1 | 6/3/22 | Ryan Reames | Updated the evaluation section with information on server side, client side, and development tool evaluations for four operating systems. |
| 1.2 | 6/19/22 | Ryan Reames | Updated recommendations section. |

**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 would like to develop a web-based game serving multiple platforms. The game will be based on their current game Draw It or Lose It, which is currently only available for Android. The game will render images which players will use to guess either a phrase, title, or thing. The Gaming Room would like to have games with one or more teams, multiple players per team, unique game and team names, and only one existing instance of the game in memory at a time.

To successfully develop this program, it will be necessary to decide the minimum internet speed required to play the game as well as who will be hosting the application. The program will need to be developed in a language that is compatible in a web environment, able to be run on multiple operating systems, and able to render images.

## [Design Constraints](#_2et92p0)

The code must be developed in a language that is not platform-specific. This is because the game will be web-based and available on multiple operating systems. Given that the game will be rendering images, the program must be able to utilize provided images. As this is a web-based application, it will be necessary to determine any limits on data, internet speed (for individuals playing the game), number of games, and total space taken by images to not exceed the host’s capacity.

To meet The Gaming Room’s requirements, the app must be able to have multiple teams with multiple players, unique game and team names, and only one instance of the game in memory at a time. This can be accomplished by the use of the singleton pattern for the game service, as well as iterators to determine whether names are unique.

## [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)

In the UML diagram, ProgramDriver runs the overall program using the other classes created for the program. The GameService class is what is used to hold the overall program, and is created using the singleton pattern, meaning only one instance of GameService can exist in memory at a given time. From the GameService class, it is possible to create anywhere from 0 to any number of Game objects as well as check for existing Game objects. By using an iterator, we can ensure that no two Game objects have the same name or ID. The GameService class is also used to generate player and team IDs.

Within the Game class, it is possible to create a Team object, and by using an iterator we can check for duplicate values. Likewise, within the Team class it is possible to create anywhere from 0 to any number of Player objects, again using an iterator to ensure that player names are unique per Team object. The Game, Team, and Player classes all inherit from the Entity class, which contains basic attributes common to all three classes. By using iterators and not having mutators once an object is created, it is not possible to change an object’s name after creation. Combined with iterators to prevent duplicates, The Gaming Room’s requirement of having no duplicate game names, duplicate teams within games, and duplicate players within teams can be achieved. Using the singleton pattern for the GameService class ensures there is only one instance of the overall game in memory, meeting another of The Gaming Room’s requirements.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Mac**  As of April 21, 2022, Apple has discontinued macOS Server, which was their dedicated server hosting software. Given the discontinued support, it would not be wise to use macOS Server for The Gaming Room’s application, given that it will not be kept up to date. They would be better off using another server platform. | **Linux**  Linux is open-source, and therefore free to use when setting up a server environment. Given that it is open source, many individuals contribute to and are constantly updating Linux. Generally, it is said to be easier for someone newer to servers to maintain a Linux server, which may be beneficial depending on who The Gaming Room will have maintain the server. If The Gaming Room would still prefer for someone else to maintain the server, plenty of server-hosting companies offer Linux as an alternative to Windows, and it still tends to be lower cost. Linux servers support Python, Perl, and JavaScript (among others) for creation, which the company could use while developing their game. | **Windows**  Microsoft offers Windows Server, which is a platform built specifically for businesses to host databases, files, applications, virtual machines, and more. Server software is kept up to date by Microsoft, who generally supports server versions for 10 years. However, the cost can add up quickly, given that a standard license costs over $1,000 and gets more expensive as more users work on the server (ex. employees). So, while overall Windows Server would likely be a good choice, The Gaming Room would need to determine a realistic budget they could stick with to use it. Windows Server uses the .NET framework, which includes the C# programming language. | **Mobile Devices**  While hosting on mobile is low-cost and there are many ways to set up such a server, this is primarily used for small, non-dynamic sites, rather than applications which will receive (presumably) thousands of visitors and requests and in-turn respond to these requests. In addition, hosting on a phone could be hazardous due to battery life, applications going into the background, and potential for loss / damage. |
| **Client Side** | **Mac**  Given that The Gaming Room is developing a web-based application for multiple platforms, there aren’t special development concerns related specifically to the Mac operating system. Where there would be concerns could be if The Gaming Room wanted to develop the application on Mac, which would be higher cost than most Windows or Linux operating systems. Beyond this, The Gaming Room would only need to ensure that clients have access to the web, therefore gaining access to the server. Developers would need experience with the programming language that The Gaming Room decides to develop the game in, and ensure that it is compatible with a web-based application. Some examples of web-friendly languages include JavaScript, Python, and Dart. | **Linux**  Similar to Mac, there aren’t necessarily specific requirements for developing for Linux, as clients will be accessing the game through the web. If The Gaming Room specifically wants to develop on Linux, they may be able to save some money on operating systems for their team, as Linux is open-source and free. And of course, the developers would need experience with the chosen language / IDE / framework that is decided upon, in order for app development to progress smoothly. Assuming developers have this experience, time would be saved as less training and research may be required. | **Windows**  Development concerns for Windows are like Mac and Linux, in that since clients are accessing the application through the web there isn’t necessarily a Windows – specific development concern. However, if The Gaming Room develops using a Windows system, they can expect a wide variety of support, given the large userbase and number of available tools for development. It would be beneficial to develop the application using a language / framework that is usable on multiple platforms, such as JavaScript with React Native Web or Dart with Flutter. This would save The Gaming Room time and money rather than developing code multiple times for the same purpose. | **Mobile Devices**  When developing for mobile devices, The Gaming Room would be best off developing the mobile app by using PCs with any of the three prior operating systems alongside languages / IDEs as described in the following development tools section. Ideally, a language would be used alongside a framework that allows cross-platform use of code, such as JavaScript with React Native Web or Dart with Flutter. Assuming this is the case, The Gaming Room should hire developers who have experience with such technology, as this would reduce time and training costs while developing the web app. By developing with a multi-platform framework, The Gaming Room would not have to develop code multiple ways to serve the same purpose, saving considerable time and money. |
| **Development Tools** | **Mac**  When developing software for Mac, a few of the languages available for use are Swift, Java, JavaScript, Python, and Dart. There are multiple IDEs available for Mac as well, including Xcode (Apple’s IDE), Visual Studio (made by Microsoft), AppCode (paid IDE from IntelliJ), and Sublime Text (somewhat more limited features than the others). Of the languages able to be coded on Mac, Swift should likely be avoided as it is specific to iOS / Mac, and can’t be used on Android or Windows. React Native Web (JavaScript) and Flutter (Dart) are two frameworks that can be used developing mobile web applications. While Xcode is free, Visual Studio and AppCode are not (at least for business), so The Gaming Room will need to determine their budget. | **Linux**  Developers can code with JavaScript, Java, Python, TypeScript, PHP, and Dart, among many other languages when developing applications for use on Linux. There are multiple free and paid IDEs available on the system, including Visual Studio, Sublime Text, IntelliJ IDEA, Eclipse, and NetBeans. Same as for Mac, two sample frameworks that would be useful in developing web applications would be React Native Web or Flutter, both of which allow multiple platforms to run code while only requiring one set of source code. | **Windows**  Similar to Linux, there are a wide variety of languages available to use when developing web applications for Windows, such as JavaScript, Java, Python, TypeScript, PHP, and Dart. There are also a wide variety of IDEs available, including Eclipse, NetBeans, Visual Studio, IntelliJ IDEA, and PyCharm. Just as was recommended for Mac and Linux, two recommended frameworks given The Gaming Room’s goals would be Reactive Native for Web or Flutter. | **Mobile Devices**  When it comes to mobile, most apps are either built for iOS, Android, or both. Common languages used are Java, Swift (for iOS), Kotlin (for Android), JavaScript, and Dart Using either Reactive Native or Flutter allows apps built to be usable on either iOS or Android. Generally, these programs are developed through use of a PC, as trying to develop on a mobile device can be inefficient when it comes to coding and working with large files. See the other categories to the left for various IDEs by operating system. |

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**:

The Gaming Room should use Linux to expand Draw It or Lose It to other computing environments. To begin with, Linux is the lowest cost solution for the company, as the operating system is free to use. It is continually updated by a large community of developers and has multiple versions which The Gaming Room can choose to suit their preferences (ex. Ubuntu or Debian). The operating system can be used for development with multiple languages that work on various operating systems (such as Dart) and can use programs such as Docker to distribute applications to multiple systems. Linux servers also tend to be easier to set up and have continuous runtime as updates can be applied while the system is running. Given that Draw It or Lose It will be a web application with a potentially large userbase, using a Linux server would be one of the best ways to get their application initially distributed for a low cost and with steady service.

1. **Operating Systems Architectures**:

Each operating platform The Gaming Room is interested in distributing their application on has a slightly different system architecture, but one thing that is common among each is the use of layering. For example, Linux layers include the kernel (core OS), shell (interface between kernel and user), and finally an applications and utilities layer. Android uses a modified version of Linux, which includes extra features designed to help memory management. Windows is layered but consists of two main pieces being the kernel and user modes. By separating the two, kernel mode can ensure users don’t access hardware without proper permissions, preventing errors. MacOS is based on Unix, and its layers include the core operating system (“Darwin”), a graphics layer, applications layer, and user interface layer. iOS is also based on Unix but has slightly different layers from macOS. iOS layers include the core OS layer, core services layer (such as storage), media layer (graphics, audio, and video), and the Cocoa Touch layer (touch-based UI).

1. **Storage Management**:

A file management system would be the best storage system to use with Linux. One reason for this is it is the default file management system included with the operating system. It is also a system that is easy to understand, as it is perhaps the most common storage management system across operating systems. Data is stored by names and can be placed in named directories, allowing the organization of data into various segments depending on business need. For example, using a filesystem for Draw It or Lose It would allow the organization of image files, program files, and servers into various directories to be accessed as needed. Another benefit to use of a filesystem is in setting file permissions. In doing so, system administrators can ensure only those with the need to access files are able to, and can set permissions to include read, write, execute, or a combination of these permissions.

Beyond this, there are a few methods The Gaming Room could implement to improve the performance and reliability of their data. First, by using SSDs rather than HHDs access to data would be considerably faster. Assuming the program does not use a huge amount of data, it would still be financially feasible purchase SSDs. To improve data security, The Gaming Room could create backups of their data, storing these on cheaper alternatives such as HDDs. Data could also be compressed when backing up to use less space. Linux uses Completely Fair Scheduler (CFS) to schedule CPU processes and Completely Fair Queue (CFQ) for I/O scheduling by default, but these can be changed to other methods such as FCFS scheduling if desired.

1. **Memory Management**:

The Linux operating system manages memory using a combination of virtual memory, demand paging, and memory caching. By using these items, the system will load Draw It or Lose It into virtual memory until required, pull only data needed to currently run the program with demand paging, and store frequently used data by caching, further reducing time required to execute frequently used items.

In addition to using Linux to manage memory for Draw It or Lose It, there are a few things The Gaming Room can do to improve memory usage. Carefully considering and choosing a programming language for Draw It or Lose It will make a significant difference, as certain languages (ex. Python) use significantly more memory than others. The Gaming Room will also want to be sure to get enough RAM for their server, as when more clients connect to the server, more memory will be required to keep up with demand.

1. **Distributed Systems and Networks**:

For Draw It or Lose It to communicate between various platforms, The Gaming Room should implement a client-server architecture. By using this distributed system, each individual client (or player in this case) communicates with the server (hosted by The Gaming Room) to run the application. The client sends requests to the server, such as starting a game, generating an image, setting a name, or logging into an account, and in response the server performs the requested action using code that is generally hosted on the server. Since Draw It or Lose It is a web application, the game will be hosted over the internet. Therefore, the server and each client’s ability to run the game will be dependent on not just the hardware’s ability to perform, but also the reliability of the connection that they have. While the application will continue to run if a client’s device fails, server failure or outage means that the application is unavailable for all clients. The Gaming Room can also set up the application so that clients are able to store some data on their computer, such as user settings stored in their browser cache. Since client-server requests are completed using HTTP, various operating systems with the ability to use various internet browsers and HTTP are able to communicate through the server.

1. **Security**:

There are various ways The Gaming Room can protect data and user information between various platforms. To begin with, when developing Draw It or Lose It the application should be developed with security in mind. For example, variables and functions that are critical to application security should be private. Beyond this, The Gaming Room should implement the principle of least privilege. This can be done by having various account types for the game (ex. guest, user, admin), each of which only have access to parts of the program that are needed by the user.

By using Linux, The Gaming Room will already have some security features built-in, such as a firewall, secure boot firmware, and other kernel security. Similar features are available on other operating systems for users, including Windows, Mac, iOS, and Android. Each of these operating systems also have password protection available, and optional security software (ex. Windows Defender). Users should be encouraged to set unique passwords for their accounts and discouraged from sharing accounts. The Gaming Room could implement minimum password requirements to increase difficulty in guessing passwords, and could also implement authentication through email or text if the user opts-in. On the server-side encryption should be used to store sensitive user data, such as emails and passwords.