

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <01/29/25> | <Aiden Villanueva> | <A first draft of the software design document for Draw It or Lose It Web Version.> |
| 1.1 | <02/04/25> | <Aiden  Villanueva> | <A second draft, editing readability and improving the Evaluation section.> |
| 1.2 | <02/18/25> | <Aiden Villanueva> | <A final draft, making revisions and applying our decisions to the Recommendations section> |

## [Executive Summary](#_heading=h.35nkun2)

Games of Draw It or Lose It will consist of one or more teams competing to guess an image that is slowly being drawn to each user’s screen. The Gaming Room aims to expand the game, currently Android-only, to a web-based version. This will:

* Expand the potential audience
* Allow users on different platforms to play together

Some critical considerations include:

* Reliably syncing the image reveal between user devices
* Keeping secure services across multiple platforms
* Managing the database of potentially thousands of images and its performance

## Requirements

The Gaming Room’s requirements for the new, web version of Draw It or Lose It include:

* Allowing multiple, cross-platform users to engage in singular matches from their own devices.
* Allow multiple teams of one or more players to assemble and cross-compete.
* Allow lobby and team names to be unique and trackable.
* Only one game instance per device is allowed.
* Game image display needs to be synced across user devices, real-time.

## [Design Constraints](#_heading=h.1ksv4uv)

Design constraints for this project are as follows:

* User platform and latency, users’ image-loading and response features must be synced accurately and processed quickly to ensure fairness
* User security and anti-cheat measures may be needed
* Scalability should be considered to allow for larger matches or team
* Cross-platform and server stability are major considerations
* Storage of the potentially thousands of images, along with their content delivery system will need special attention
* User interfaces and UX will need to be adjusted per platform for accessibility and reliability

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

The following UML diagram shows the overall package layout for the Draw It or Lose It game. The diagram shows the main Program Driver and a Singleton Tester to run and debug the program. Within the game classes, we can see an Entity base class that is inherited by the classes Game, Team and Player, giving them all basic attributes like id and name, as well as some basic constructors and mutator/accessors, and a function to turn an argument into a string. Lasty, we see the GameService class, of which we know there should only be one at a time, per device. This GameService manages the game state as well as all networking and game logic. This class is also created in a singleton pattern to prevent multiple instances, by passing a reference to itself through the getInstance() method, which is public and can be called by any other class in the package. The Game entity is a container object for storing information about the teams and current game. Game containers will also instantiate zero or many Teams, which in turn create none or many Players, both of which are also data container classes.

**"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](#_heading=h.z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs can pose a problem in the way of networking, as stricter protocols and proprietary software can become cumbersome. Macs have the added advantage of being very secure. | Linux’s main challenge will be ensuring consistency across many distributions and hardware setups. Linux has the advantage of complete control over hosting processes. | Windows has a fairly simple and safe networking protocol to build upon, making it the easiest of the bunch. Windows has the advantage of being easy to use and implement. However, without intervention, Windows is the least secure of the bunch. | As Android is already supported in the old version, adapting the game to include iOS and retaining Android should be straightforward. Some specialized libraries may be needed to include iOS in the project. |
| **Client Side** | Macs may require one-time or persistent licenses to host and operate the app’s client side services (app, domain). May be costly. Mac hardware is reliable and usually powerful enough to not hinder performance. | Linux, being open-source by nature, will have a free rollout, outside of the labor hours needed for the previously mentioned issue. Further, user support can be difficult given the diversity of the audience. | Windows has flexibility in choosing a simpler, more expensive route with outsourced hosting, or a centralized, local version. Windows can potentially be more vulnerable to malicious behaviours, especially when using file sources like images - this furthers the need for a server-client architecture. | Android publishing will need a Google Developer account, if none exist, and iOS will need an Apple developer account and iOS test devices or emulators. While mobile devices are fairly safe automatically, additional security can be employed when handling sensitive user data. |
| **Development Tools** | To deploy and test on Mac, while maintaining only one codebase for the project, I recommend using a cross-platform web library such as React or Vue, and make the project run off of HTML, JS, and CSS for compatibility. | If the simpler route is not an option, Java can present another possible main language, while using cross platform libraries like Node.js or Xamarin, as Java is supported natively in most Linux distros. | Windows has a wide range of support for programming languages, tools and libraries, making it the least worrisome and most adaptable. Recommended to use | While Android is built around Java, it has its own runtime, and therefore needs its own libraries. To include Android and iOS in our platform catalogue, we should utilize React Native - this allows a single Java codebase, with cross-platform usage and deployment. |

## 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 recommend using Windows to develop Draw It or Lose It, as it offers the most compatibility inside a stable and user-friendly workflow environment. It aligns well with the goal of using Java and a framework like React Native for deploying a web-based app to multiple platforms. For hosting a server side applet, I recommend using AWS cloud servers along with Windows or Linux to run the server app. Meanwhile, each user will be able to choose their platform from Windows, MacOS, Android, Linux and iOS.
2. **Operating Systems Architectures**: Windows, Mac, Linux, Android and iOS can all be included as exports if we use a development framework such as React Native - which allows cross-platform exports, namely to iOS. The client app can be written in Java for its many cross-platform libraries and features, while being fairly fast and type safe. The server-side app can also be written in Java for simple compatibility, or in a lower level language like C++ to give more control over server features like security and memory management.
3. **Storage Management**: I suggest utilizing cloud storage for the image streaming functionality and server side app. This enables rapid scalability and quick iterative deployment, while allowing for fallback states for 100% uptime. The client app will buffer, cache and save images as they are loaded from the server during play, and cleanly remove images from RAM after being used in a round. However, the user’s device will store the images for fetching later, and if available, will use the local copy instead of re-downloading. This should be coupled with a minimal system to check image files for corruption and re-download if needed.
4. **Memory Management**: Windows has a multitude of tools to track and debug memory management, as well as enabling the testing team to use emulators of other platforms inside of Windows to test memory usage and test cases inside of those deployments. This makes it a great candidate for the development environment for Draw It or Lose It. APIs will be used where necessary to facilitate platform-specific memory management techniques.
5. **Distributed Systems and Networks**: Since Draw It or Lose It needs to connect users of many platforms together, I suggest utilizing a Server entity backend to host the game lobby over the internet and manage player global syncing as well as the image relaying. This Server can exist on a cloud server, but local play over WAN or LAN networks should be included as a fallback option. Hosting a single server allows for communication code to be confined to one codespace, while the user-end functionality can be tweaked per-platform, when needed. By hosting several cloud servers at once, this provides redundancy, scalability and user choice.
6. **Security**: By separating the client user experience from the running data of the game on the server, we provide the first layer of security to the app. Additionally, we should encrypt user data like names and ip addresses, and ensure that the system for sending packets to the client system is secure to external sources and encrypted. To maintain playability and reduce UX clutter, these features should be mostly automated and silent.