

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.30j0zll)1

[**Table of Contents**](#_heading=h.1fob9te)2

[**Document Revision History**](#_heading=h.2et92p0)2

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

[**Design Constraints**](#_heading=h.3dy6vkm)3

[**System Architecture View**](#_heading=h.1t3h5sf)3

[**Domain Model**](#_heading=h.4d34og8)3

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

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.2et92p0)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/13/2021 | Lawrence A. | Version 1.0 initial design presentation |
| 1.1 | 07/31/2021 | Lawrence A. | Evaluation |

**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](#_heading=h.tyjcwt)

“Draw it or Lose it”, the new game from “The Gaming Room”, will begin development soon! This unique web-based game will see players from different teams competing in an online game environment to see who can guess the puzzle first. At CTS, we will begin to develop the core engine that will run the game for users. The Gaming Room requests that players be able to join teams with unique names, and that games be given unique identifiers to better organize the players and teams. Additionally, only one instance of the game engine should ever be running at once; this will more than likely save on bandwidth and minimize memory leaks, leading to a smooth and consistent gaming experience for all users across all platforms.

Taking these under consideration, it is advisable that the game engine utilize the singleton design pattern to ensure only one instance of the game engine is running at any given time. Within this game engine, we will develop the core objects of the game: players, teams, and games themselves. The game engine will be able to run multiple games at once, and each game will consist of multiple teams. Each team can have between 1 and (4) players at a time, and each will be assigned a unique identifier. It is also advised that each object be a

## [Design Constraints](#_heading=h.3dy6vkm)

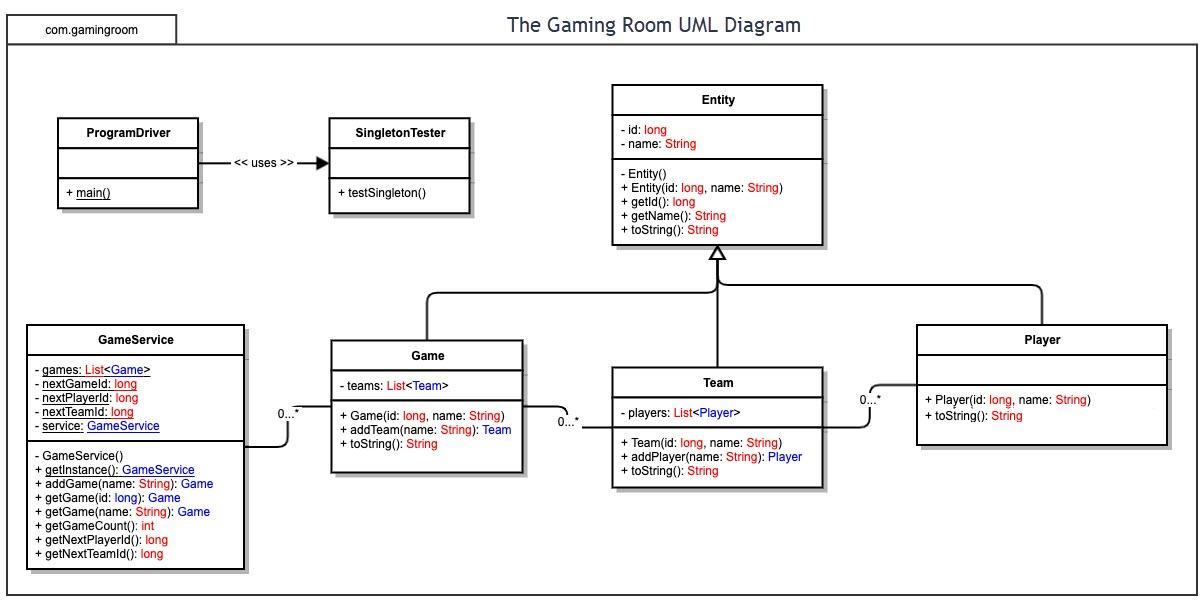
Potential customers may use different web browsers, different computers, different internet connection speeds, or any combination of all three. Taking this into consideration, software should be developed with a “minimum system requirements” base in mind. This will affect the development so that more complex features that use resources not available to every customer/player may need to be cut or at least minimized in scope or implementation. Different web browsers should also be considered during development, since this can be akin to different operating systems on a PC.

## [System Architecture View](#_heading=h.1t3h5sf)

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.4d34og8)

Based on the diagram below, engineers can see that all three classes (Game, Team, and Player) fall under the umbrella class “Entity”. This allows all classes to “talk to” one another, and maintains symmetry in the code, as each class will be required to have the same base variables as described in the Entity class. Each class will contain an ArrayList of the class “below” it. Each object of the class will contain it’s own list of lesser objects; an object of type “Game” will contain a list of all “Team” objects associated with that game, and each “Team” object will contain it’s own list of “Player” objects on that team. All “Lists” will be accessible through the GameService class, which will act as a mediator from the Program Driver (main) class.

****

## [Evaluation](#_heading=h.2s8eyo1)

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 OS Server is closed-source and mainly for smaller projects, like home servers, and would not fit the scope of this project. | Variety of options to choose from, each with different important features highlighted. Developers will need to be proficient in CL. No upfront costs involved. | Increase in users means an increase in licensing costs. Uses ASP.NET, a high performance web app framework not found in Linux servers. Can be more vulnerable to malware. | Not applicable; no mobile devices will be used to host this game, as they will not provide the power nor storage necessary for large deployment. |
| **Client Side** | Since a REST development design is being used and all client-side code is being delivered via HTML, a working web browser will be the only requirement of the user. Each OS listed typically has their own built in browser (Safari, Firefox ESR, Edge) so these should be top priority for compatibility (security updates for the current browser should be the first compatibility test). After these, the top browsers that users are likely to use (Chrome, Firefox, Opera, etc) should be tested for compatibility.  Desktop browser size and shape can be adjusted on the fly by users, so ratio and display compatibility need to also be considered for each of these operating systems. A minimum aspect ratio and display density should be determined for the app to run properly, as well as upscaling for higher resolution displays. | | | Mobile devices will have built in browsers that should be tested for compatibility, as well as third-party browsers that users may use. Adjusting the app for various aspect ratios and display resolutions should be a priority. Additionally, considering portrait versus landscape and if a user should be able to switch will be imperative, since users may have a preference. |
| **Development Tools** | Developing this app in Java will require a complex IDE like Eclipse (free), and a JAR bundler to create to turn the class and other associated files into a Mac OS application. This can typically be done on a Mac from the command line, or there are various free tools available to streamline this process. Client computer will need java installed. | Linux can be used for both development and deployment of the app. There are various free IDE’s for developing the app. Using Maven, the jar file can be converted to a DEB file (for use on Linux machines) by adding certain scripts to the pom file. This will allow installation on Linux-based machines. | Again, Eclipse is the right way to go here for development of the Java application, and for creation of the .exe file we will want to use maven again (with a certain script added to the pom file). The .exe file can be distributed and installed on Windows machines. | IOS: xCode  Android: Android Studio  Java apps will run on both platforms. However, for iOS it is more acceptable to develop using Swift in xCode if possible. Both Android Studio and xCode are free to use on their respective platforms (Windows and Mac). |
| Developing an app for each individual OS would be time and resource consuming, needing developers who are experienced in working with each OS. Consolidating development into one language and one deployment type would save on testing and developer costs, as well as maintenance costs (security and app updates). If the app is to be developed for initial deployment on a server and then distributed via HTML to web browsers, then developing on one desktop OS using Eclipse is advisable. This will shift developer focus to streamlining server-client connections on all bandwidth speeds by developing the app to minimize data to only necessary files needed at any given time. Testing will focus on this aspect as well, specifically various connection types and speeds.  Multiple development teams will focus on each main aspect of the app: client side (game), server side (control), and lobby (anything outside the app but still user accessible, like friends lists and main menus). Solid communication between teams will be necessary so there is constant balance in the development process. If individual apps are to be developed for each OS, the same team can handle the conversion of the apps to run on each OS.  Majority of costs involved will be in paying developers first (a team of 5 to 7 should be sufficient, depending on the time frame), then server costs for hosting the app (whether it be deployed for install on individual systems, or hosted as a REST design). | | | |

## 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**: For Draw It or Lose It to run on the client servers, Linux is the recommended operating system. Linux, being open source, gets constant updates to maintain security and operating standards. These frequent updates will help keep the server running smoothly and securely for users of the game. Linux can also be customized more than other OS’s, with either specific versions and distros, or under the hood. This customization will allow the client to tweek certain system aspects to improve speed and connectivity, so that users can get the best experience when playing the game online. Most Linux distros are free, which will ultimately reduce client costs in the long term (although this means that tech support is minimal, unless a specific distro is purchased and support is included). Linux also does not require regular reboots, whether from software installation or normal usage slowdown; this will allow the server to maintain optimal speed in application execution all day and night long.
2. **Operating Systems Architectures**: Linux architecture is similar to other OS’s in that, at its core, consists of a kernel, wrapped in a shell that facilitates user interaction. Many distros of Linux have two options for accessing the kernel: GUI and CLI. For the Gaming Room’s purposes, a simple CLI is all that will be needed, as this will reduce the amount of overhead for the server hardware to deal with overall.
3. **Storage Management**: The recommended storage hierarchy for Draw It or Lose It should be a hybrid type (if all Solid State storage is too expensive). Maintaining the application on SSD storage would be beneficial, but not necessary. The application files will be copied to memory, so storing these files on slower platter-drives is fine. Pictures that need to be accessed quickly throughout the game should be stored on faster SSD’s, and user info, which will be accessed only once at the beginning of the game session, can be stored on platter-drives as well. Storing files on the hard drives should be done in a combination of contiguous and indexed allocation methods. This will allow for quick random access to picture files, but also minimize wasted space. The application data files will not be changed very often, but it will be good to have them stored in a manner that doesn’t fragment the files anyways, since they still need to be accessed regularly.
4. **Memory Management**: Maintaining memory during gameplay will be key to ensuring a smooth experience for the users, so it is recommended that developers take advantage of Java’s built in “garbage cleanup” commands to remove unused items from system memory. Loading only key data into memory at any time and utilizing the garbage cleanup will keep memory usage low, and allow for less memory modules overall, again cutting costs. Refining the code and removing redundancies will also reduce the file sizes that need to be loaded into the memory at game startup.
5. **Distributed Systems and Networks**: As has been mentioned before, with a distributed software model, the game Draw It or Lose It can be accessed by multiple users from various platforms via one generic type of server connected to the internet. That is, the Gaming Room’s servers can house all necessary data, and send the requested files to users’ internet browsers for rendering of game graphics and game data.

For PC users, a simple web browser will be sufficient, as all data will be sent via html when accessed by users. For mobile clients, individual apps for each mobile OS can be developed to take advantage of the given OS’s built in web browser. Since all users will access the servers the same way, this will minimize how many versions of the game must be developed. Some important files may be downloaded to the user’s device in order to save on bandwidth (such as large sized graphics), but these too will be accessed via commands from the server during gameplay.

To maintain the best speeds during online gaming, the client should consider a very fast speed connection from the server to the internet. User speeds will vary, but client-side speeds should be as fast as possible to minimize bottle-necking during multiple gaming sessions as data is transferred around the internet. Downloading larger files to user devices could also be considered in order to keep gaming speeds the same across all connection types.

1. **Security**: Security will be built into the app itself, with user authentication methods employed at the start of each game session. There are various services that offer this option via Java code, such as dropwizard, that seamlessly integrate authentication into the main app’s code. Developers and testers will monitor the code for vulnerabilities, and update as necessary.

The Linux OS on the server is the least vulnerable of all OS’s available, requiring password input for program execution. There are very few vulnerabilities built into Linux, and constant updates ensure that any new vulnerabilities are dealt with swiftly.