

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[CS 230 Project Software Design Template 1](#_Toc2026853699)

[Table of Contents 1](#_Toc1530991178)

[Document Revision History 2](#_Toc1220986307)

[Executive Summary 2](#_Toc1705735667)

[Requirements 3](#_Toc2794183)

[Design Constraints 3](#_Toc2047696364)

[System Architecture View 3](#_Toc918338551)

[Domain Model 3](#_Toc1878604955)

[Evaluation 4](#_Toc1989843822)

[Recommendations 6](#_Toc892924904)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/24/2025 | Lance Tran | Initial release |
| 1.1 | 08/08/2025 | Lance Tran | Revisions to Evaluation, recommendations. |
| 1.2 | 08/22/2025 | Lance Tran | Revision to 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 client, The Gaming Room, has requested assistance in creating a design document to facilitate the development of a web-based application called “Draw It or Lose It.” The game itself is based on a game show, Win, Lose, or Draw, where teams compete to guess what is being drawn.

The purpose of this document is to: Determine the requirements, the prerequisites, constraints, provide a general layout of the design, and offer an overview of various strengths and weaknesses of platforms the application will be on. By the end, the reader will understand the goal and the intended means of achieving it.

## Requirements

1. Users should be able to log into the game from any platform and join the same servers. That is partially achieved by being web based.

2. The game should be able to manage multiple teams made of one or more players.

3. Game instances need to be managed, starting instances as needed and removing them when the game instance is concluded.

4. Game instance and user names should be unique to avoid conflict

## [Design Constraints](#_2et92p0)

1. Web-based Distributed Environment: The game application must be cross-platform compatible; therefore, a constraint is to maintain compatibility.
2. Every user should have a unique username and ID
3. Every user should be limited to one game instance at a time

## [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)

"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.

Above is the diagram for how the application will operate. The Entity is the superclass that provides common traits for the subclasses such as Game, Team and Player to inherit. Those subclasses extends the Entity superclass in various ways. The game class adds a team, team class adds a player, and the player class has a constructor that is used to create instances of Player. The GameService class manages lists of games, but it is unique in that it does not inherit anything from the superclass Entity.

The relationship between GameService, Game, Team and Player is that they are in aggregation relationships. This is different from a composition relationship because there is no strong ownership between classes. For instance, a player can be initialized from the player class, the Team class simply lists the Players that have been initialized, Game holds a list of teams that were initialized, and GameService holds a list of games that were initialized.

ProgramDriver is the class that contains the main() function which calls the testSingleton() function of the SingletonTester class. Only one GameService instance can exist at a time in a proper singleton arrangement. After the testSingleton() is called, main will include code that interacts with other classes.  
Because of their arrangement, they are dependent on each other, using each other, so to speak.

There are a number of object-oriented programming principles demonstrated in this UML diagram. Inheritance is a concept that is demonstrated by the fact that the classes Player, Team and Game inherit common traits from the superclass Entity. An example of encapsulation is that classes have private attributes to protect access of the class exempt through specific code. Abstraction occurs partially because of encapsulation, when classes have private attributes it makes sense to hide them, and when you hide them, you focus more on what is done, rather than how. It becomes a “black box” where you know if you do A, then B happens.

## [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 must work together.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Unix-based  Database – SQlite, Postgre SQL  Server – Apache, Nginex  Moderate hardware requirements | Server – Apache, Nginex  Database - MYSQL  Lowest requirements for server hardware, easy to run “headless” | A Windows server OS  .NET framework for server applications  Web Server - IIS  Database – MS SQL  More hardware intensive than Linux servers | Mobile devices require efficient API usage the most to minimize data transfer and latency to provide the best user experience to the user. |
| **Client Side** | Strong user experience through interface,  But applications can be rejected if they do not meet Apple’s UI standards. | Can be less intuitive or harder to install and work properly compared to other operating systems. May require installation of dependencies, or obscure drivers. | Most of the PC audience that play games use Windows.  Familiar for most to learn how to access and play.  Distribution may require licensing and more costs. | While it is easy to distribute on Android, iOS is known for being strict on qualifying to be distributed. |
| **Development Tools** | Not as many development tools and few are cross-platform. Xcode is notable. | Lots of IDE options and a powerful command shell, options for package management for distribution | C# and .NET frameworks are used Examples of IDEs are Visual Studio and JetBrains. | Swift, Objective-C, Java and JavaScript. Android Studio and XCode. Many PC browsers can test web applications in a simulated mobile mode. |

## 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 recommended operating platform for the servers would be a Linux platform. It is known as a cost-effective platform for servers as there are no licensing fees. It’s open source, flexible, scalable, and has lots of options for security. The operating system is open source so transparency is strong. There are several different distributions to be more fitting to the exact needs of the server. A GUI is not necessary; it is easy to roll out updates remotely by command line and remain secure. It has a steeper learning curve, so retaining skilled employees to maintain it is a consideration.

2. Operating Systems Architectures:

I have flip-flopped on MPA vs. SPA quite a lot, and I am back on team SPA.

SPA best mimics the app-like user experience regardless of platform, not requiring full page reloads. It might take a bit longer to load initially as it would load the entire necessary framework, but afterwards, it will only upload necessary parts. The intent is to support the idea that the lobby of a game will be its own page. Once you’re in a lobby, REST API to allow numerous calls to be made, and instructions from the server can be received from the server. The downside of this is the natural weakness of security issues, but it can be solved by strong server side data validation, and limiting open API endpoints. In other words, assiduously designing the program easily mitigates its flaws.

3. Storage Management: Because Draw It or Lose It will have a fixed number of images that will be drawn, it makes sense for most of the images to be pre-downloaded and stored locally on the client. Instead of transmitting the images to players in real time, the server will only send a cue to the client to proceed with the display. Because it’s only sending a code instead of an image, the overall latency can be minimized. Therefore, a server for initial download of assets and then local storage usage is recommended. The downside is that it will be less portable, compared to all the images being pulled from server and managed as needed. In the future when more images are added, a slight compromise could be introduced by having images in sets as “rotations” during “seasons.”

4. Memory Management: Initially, I believed just deferring to the browser would be the way, but it would be too optimistic and leaving too much in the hands of others. Memory leaks can still occur regardless of browsers handling garbage collection. The program should still be designed in a way that large data structures are still managed, and unused allocations are deallocated. Optimize and decide how to handle frequently created and destroyed game objects. For example, if you know you’re going to show a text object frequently, is it better to delete it, or to hide it and show it again later? For a game like Draw It Or Lose It, it is possible to make most assets and text objects premade and called to screen. When working with a set pool of assets, it’s easier to keep track of things assigned to memory and not increase usage over time. A common cause of an increase in memory usage is that even if an object is no longer being used, it still might be referenced to, preventing garbage collection from addressing it. Pooling objects helps mitigate that possibility.

5. Distributed Systems and Networks: To enable communication between various platforms, the

game can utilize distributed software architecture and leverage network connectivity. This can

be achieved by implementing a centralized server or cloud infrastructure that serves as a

communication hub between game clients. The server can handle game synchronization, real-

time updates and message exchange between players across different devices. The system

should account for network connectivity issues, such as intermittent outages or low bandwidth,

by having appropriate error handling and synchronization mechanism.

6. Security: User information and the transmission of private information needs to be protected. As a web-based application using correct secure communication protocols is important. As a “for fun” game, hopefully player integrity is not an issue, but if it does become one then somehow the choice of which image is selected needs to be encrypted or otherwise obfuscated from unauthorized access or viewing.