

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/23/2025 | Mitchell Flint | Completed sections: Executive Summary, Design Constraints, and Domain Model. |
| 1.1 | 04/6/2025 | Mitchell Flint | Completed section: Evaluation. |
| 1.2 | 04/14/2025 | Mitchell Flint | Complete section: 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)

Draw It or Lose It is a game inspired by the 80s TV game Win, Lose, or Draw. In it, teams compete to guess the solution to a puzzle, based on image clues. The Gaming Room needs a solution for supporting multiplayer functionality, by providing unique identifiers for games and teams. This can be accomplished by have a singleton object handle the ID numbers themselves.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Making Draw It or Lose It a web-based game comes with some important design constraints. The game needs to be able to make sure each game, team, and player has a unique ID. Since only one game instance can be active in memory, the system has to manage that carefully, using a singleton setup to ensure there are no duplicate IDs, and that game logic remain synchronized among each player.

## [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 UMP diagram shows how the game is organized using principles of object-oriented programming. The Entity class is a base class that holds shared info, and it’s extended by the Game, Team, and Player classes, in order to avoid repeated code. A Game can have several Teams, and each Team can have several Players. The GameService class uses the singleton pattern, so only one version of it runs at a time, which fits the requirements of having just one active game in memory. It keeps track of all games and generates unique IDs. The OOP of inheritance, encapsulation, and clean structure makes the app easier to manage.

**"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)

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** | macOS can be used to host web applications with tools like Apache or Nginx, but it’s not typically used for large scale hosting. It requires Apple hardware, which can be costly and less scalable. There are no licensing fees for the OS itself. | Linux is the most common platform for web servers. It supports tools like Apache, Nginx, and Docker, is highly scalable, and doesn’t require a license, making it cost-effective for server deployment. | Windows Server works well for web hosting, especially with .NET-based applications. It supports IIS, but there are licensing costs, and it may require more configuration to scale effectively. | Mobile devices are intended for users, not for hosting. They can’t run a reliable server for large numbers of players, so this option isn’t practical for the server side. |
| **Client Side** | The game should be tested for compatibility with Safari on macOS. Developers need Mac hardware for this, which can increase cost or slow development. However, macOS users expect polished, responsive web experiences, so supporting it is still valuable. | The game should support browsers like Firefox and Chrome, which are common on Linux. Development and testing are straightforward, though the desktop Linux user base is relatively small. | Windows has the largest desktop market share, so strong compatibility with Chrome, Edge, and Firefox is important. Development is well supported with popular tools and frameworks. | The game needs to be responsive and work in mobile browsers—Chrome for Android and Safari for iOS. Development takes extra effort for cross-device testing, but frameworks like React or Flutter can help streamline the process. |
| **Development Tools** | macOS supports Xcode, VS Code, and JetBrains tools. These are mostly free, though building for iOS requires Apple hardware. Teams may need at least one Mac for testing and deployment. | Linux offers a wide range of free tools, including VS Code, Eclipse, and command-line utilities. It's excellent for backend development and doesn't involve licensing fees, though it may require more technical familiarity. | Windows supports Visual Studio, JetBrains Rider, and many other tools. It’s developer-friendly, and while some tools require paid licenses, there are plenty of free options. Widely used in both frontend and backend development. | Mobile development may require Android Studio and Xcode for native apps, but since the client wants a web-based solution, using cross-platform frameworks (like React) is ideal. Most tools are free, but building for iOS still requires macOS. |

## 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 the server-side platform, I believe Linux would be the best fit for hosting Draw It or Lose It. Not only is it widely used for web applications because it’s stable, but it also doesn’t come with licensing fees. It integrates well with cloud providers, which would allow the game to scale easily and handle a large number of players without being limited by physical hardware. Since the game needs to run continuously and respond to users in real time, Linux prioritizes performance and flexibility.
2. **Operating Systems Architectures**: Linux has a modular architecture that supports multitasking, process isolation, and efficient resource management. It supports container based deployment, making it easy to separate core services like game logic, matchmaking, and user sessions.
3. **Storage Management**: As for storage, The Linux server uses a modern and reliable file system designed for stability and performance in web-hosting environments, which adds flexibility in managing disk space. Game assets and backups can be stored using Google Cloud Storage.
4. **Memory Management**: Memory management is another strength of Linux. It uses paging, copy-on-write, and virtual memory to make sure resources are used efficiently. This is especially important when multiple services are running at once. It includes tools that would let us control how memory is allocated, so that the system can remain responsive during peak hours.
5. **Distributed Systems and Networks**: Since the game needs to support players on different devices, it should be built as a distributed system, with each backend element containerized: login, game logic, and score tracking should each run independently across multiple servers. These services stay in sync by exchanging data through network requests, allowing players on mobile, desktop, or consoles to interact with the same game sessions. By incorporating redundancy into the system, when one part of the system goes down, the activity can be shifted to the other servers so that the game session can continue without interruption. This setup also allows the game to scale more easily as more players join, in order to maintain performance, even during peak hours.
6. **Security**: Linux provides the tools needed to handle security as well. Its servers can implement HTTPS to protect data in transit, and it can handle user authentication with token-based systems to prevent unauthorized access. It supports these standards with built-in SSL/TLS tools, strong file permissions, and integration with PAM, allowing compatibility with external login services. Firewalls and regular updates help limit vulnerabilities, which works well for protecting the user’s data from breaches.