

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 3.0

## 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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 5/24/2025 | Rebekah Frank | Created document for draw it or lose it application. |
| 2.0 | 6/7/2025 | Rebekah Frank | Updated and revised document to match specifications related to server side, client side and development  tools. |
| 3.0 | 6/20/2025 | Rebekah Frank | Made final revisions to chosen recommended OS for game. |

## [Executive Summary](#_sbfa50wo7nsh)

The client, The Gaming Room, wants to create a web-based application game titled “Draw it or Lose it.” The game is loosely based on the television game *Win, Lose or Draw* where teams try to guess what’s being drawn. The application will render photos from a large library of stock drawings to be used as clues where each drawing is rendered at a steady rate and are fully complete by the 30-second mark. There will be four rounds, with each round lasting one minute. If a question is left unanswered, the opposing team will have 15 seconds to correctly guess. Expanding the program to a web-based application will allow more devices and platforms to be able to play the game, which ultimately enhances user experience.

**Requirements**

*•* The game must be translated into a web-based application, allowing for wider access across devices.

• Naming of the game and teams must be unique to avoid any conflict within the program.

• Teams should have multiple players assigned for ideal gameplay conditions.

• Each game will have the option to involve one or more teams.

• The game must check and ensure only one instance of the game is running at a time.

## [Design Constraints](#_2et92p0)

• Web-based application

* Translating the game into a web-based application from an android app introduces a new set of challenges, which include ensuring compatibility, security conflict, etc.

• Single instance

* Ensure only one instance of a game can exist at a given time to avoid memory misuse and ensure the program runs smoothly.

• Unique naming conventions

* Ensuring the names of the teams and players is unique is crucial. Overlap of names and players could cause errors in the program which would diminish user experience.

## [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 diagram below depicts the classes and relationships in the *Draw it or Lose it* application. The Entity class purpose is as a superclass to the Game, Team, and Player classes, where each subclass inherits attributes such as id and name from the Entity superclass. A game can include multiple teams, and each team includes multiple players.

The GameService class manages Game instances and is composed of multiple Game objects. Similarly, Game contains Teams, and Teams contain Players. The ProgramDriver class creates a singleton instance of GameService to ensure only one instance of a game is in memory at a time. The SingletonTester class tests to make sure only one instance does indeed exist at a time.

The UML diagram reflects core principles such as inheritance (Entity and its subclasses), encapsulation (private attributes and controlled access in GameService), and abstraction (simplifying interactions between objects while hiding internal complexity).

**"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** | Unix based, which can be reliable and developer friendly. Can have higher hardware costs and reduced scalability compared to other OS’s. Usually is more difficult to scale than other platforms, making it costlier and more time consuming to scale. Server licenses can vary from around $25-100+ a month, additionally, an apple developer membership is required to publish to Apple devices costing an addition $99 a year. | Open source and cost friendly. Flexible, stable, and scalable, but may have GUI and hardware compatibility issues. May require additional setup time or a need for specialized support. Usually low cost ranging from around $15-50+ a month. | Widely used OS, which makes it highly compatible and developer-friendly, but prone to more security vulnerabilities and malware. May require additional tools to keep secure, increasing to the cost and development time to integrate. Depending on how the app is hosted, the cost can range from around $30-200+ a month. | Easily transportable. Uses touch on small screens. Has varying hardware that affects performance and user experience. Keeping in mind the overall diversity of mobile devices needs to be considered and may add to the overall cost of the project. |
| **Client Side** | Simple and user-friendly interface reduces the learning curve. Developing and supporting multiple clients may require time, increases costs, and necessitates a broader range of technical skills required. | Free to use and share, however not as widely used as other OS’s. May involve additional costs and a steep learning curve for new users, requiring varied expertise for different clients. | Widely used as a common operating system, clients will be more familiar with. Has a wide array of support. However, licensing expenses can exceed those of open-source options. | Provides remote access flexibility to clients, but with higher implementation complexity and key challenges like responsive design, connectivity, and native feature integration. |
| **Development Tools** | Node.js and  JavaScript are commonly used. Common IDE’s are:  Xcode (Apple's official IDE for native apps) and VS Code (supports web and cross-platform development.)  While most of these are free to use, Apples IDE (Xcode) does require an apple developer program membership at $99 a year in order to publish to the app store. | VS Code (lightweight and extensible), Vim/Neovim (customizable terminal editors), Emacs (powerful and scriptable) The IDE’s are free to use and require no fee’s. | C# with Visual Studio is commonly used in Windows app development, C++, and JavaScript/TypeScript (native and cross-platform apps.) VS Code (lightweight and extensible) While all of these are fundamentally free, Visual Studio does offer a paid version in both professional and enterprise, costing 45 and 250 a month respectively. | Apple uses swift for IOS development. Android mainly uses Kotlin. Also uses a mix of other languages such as: C/C++, Java, Javascript, etc. Most of these are free, however Apple still requires a $99 a year fee to publish on the app store. |

## 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 ideal operating platform for translating The Gaming Room’s *Draw It or Lose It* to other computing environments is Windows. It offers the greatest flexibility, with a wide range of IDEs and development tools at the ready. Because Windows is a widely used operating system, it will be accessible to a larger number of users.
2. **Operating Systems Architectures:** Windows is an operating system developed and published by Microsoft, designed to provide a user-friendly interface for managing files, running software, playing games, watching videos, and connecting to the internet. It is one of the most versatile and widely used operating systems globally.
3. **Storage Management**: Windows storage management provides tools like Disk Management and Storage Spaces to organize data, using NTFS for secure storage, along with features such as Disk Cleanup and Storage Sense to optimize disk space and performance.
4. **Memory Management**: The web-based platform uses browser garbage collection for automatic memory management, improving efficiency and reducing memory leaks. Meanwhile, Windows Storage Sense securely organizes and manages Draw It or Lose It photos and player data in one central location. On top of that, Windows manages memory for Draw It or Lose It by using virtual memory, paging, and tools like Memory Compression and SuperFetch. These help keep the game running smoothly by prioritizing resources and reducing slowdowns.
5. **Distributed Systems and Networks**: Network games rely on a shared database for players to interact remotely, however developers often build this system from scratch. Using a centralized server or cloud-based architecture can streamline communication, manage synchronization and updates, and handle network issues to support cross-platform gameplay. One detail to keep in mind is that interdependencies among components, such as server availability, internet connectivity, and client responsiveness must be accounted for to ensure seamless cross-platform gameplay.
6. **Security**: Protecting user information requires security measures including secure communication protocols (such as HTTPS and SSL/TLS), strong user authentication (e.g., multi-factor authentication, biometrics), and end-to-end data encryption. Although Windows has built-in security, additional measures should be taken for stronger data protection. These may include intrusion detection systems (IDS), regular security audits, access control policies, and real-time threat monitoring. Implementing regular updates and general user education can also play a major role in protecting and keeping a secure environment.