

Gaming Room, DRAW IT OR LOSE IT

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 08/12/2024 | Mohammed Alshehabi | Have redone this from scratch to make sure that Turnitin score is under the usual standards.  Wrote down info for the evaluation section |

**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)

I am writing this summary to present the design plan for the web-based game "Draw It or Lose It," which is based on the popular TV game show "Win, Lose, or Draw," the game is about teams guessing images that are drawn. The proposed solution focuses on expanding the game's reach by making it available across various devices via a web platform. The design goal is to meet the client's needs, making it an easy and smooth, user-friendly gaming experience that support multiple teams and rounds of gameplay.

## Requirements

*-****Cross-Platform Accessibility****: The game must be accessible through web , and be compatible with different devices and operating systems.*

*-****Team-Based Gameplay****: The game has to support more than one team, with multiple players, making sure the player have unique names and experiences for every user.*

*-****Unique Identifiers****: To prevent problems, every game, player, and team must have special identifiers.*

***-Memory Management****: The app got to make sure that only one instance of the game is active in memory at any time, using unique identifiers for games, teams, and players.*

*-****Timed Rounds****: Each round got to last one minute, with drawings smoothly presented until the 30-second period is reached. If maybe the puzzle is not guessed in time, other teams can have a chance to guess but it has to under 15 seconds.*

## [Design Constraints](#_2et92p0)

-**Web-Based Environment**: This game going to developed for a web-based platform, keep in mind network communication, security, and cross-browser compatibility.

**-Name Uniqueness**: The system must make sure to have special names for games, teams, and players to not have any trouble in the future and improve the end user interactions.

**-Single Instance Restriction**: The design must make sure that only one instance of the game service is active in memory at any 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 diagram shows the main parts of the "Draw It or Lose It" game and how they connect. At the Middle is the Entity class, it acts as a base for other parts of the game. This class has common features like id and name, which are shared by all other parts of the game.

There are three main parts that come from the Entity class: Game, Team, and Player. The Game class is in charge of a group of teams (represented as a list of Team objects). In the same way, the Team class is responsible for a group of players (represented as a list of Player objects).

The GameService class is very important because it manages the games. It uses a special design called the singleton pattern, which makes sure that only one instance of GameService exists while the application is running. This class helps create and manage games, teams, and players. It keeps track of all the active games and has methods for adding new games and getting details about teams and players.

The ProgramDriver class is like the starting point of the game. It sets up the game environment and uses the features provided by GameService. It also checks that only one GameService is running, with help from the SingletonTester class.

**"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** | Linux is a stable and flexible operating system that’s great for running big web applications like "Draw It or Lose It." It’s free to use, which helps keep costs down as the game grows. Linux is also known for being secure and customizable. But, it can be hard to learn. | MacOS is built on a strong Unix base and is known for being easy to use, especially for developers who are familiar with Apple products. It works well with other Apple devices and offers powerful tools like Xcode. But, MacOS comes with higher hardware costs and doesn’t scale as easy as Linux, which might be an issue for larger projects. | Windows is widely used and works well with a lot of different software and hardware, making it a good choice. It’s familiar to many developers and integrates well with Microsoft tools like Visual Studio. On the downside, Windows has more security issues than Linux and MacOS, so you might need to spend more time and resources to keep it secure. | Android is very flexible and runs on many different devices, making it great for reaching a big audience. iOS is known for being smooth and secure, especially on Apple devices. However, Android's variety of devices means the app needs to work well on many different screens and hardware, which can be tricky. For iOS, developing can be more expensive because it requires Apple tools and devices. Both platforms need servers that can handle different internet speeds and connections, which can be tough, especially since mobile users often have spotty connections. Keeping everything running smoothly on both Android and iOS can also take more time and cost more money. |
| **Client Side** | Linux on the client side is free and highly customizable, making it a great option for users who like to tweak their systems. It supports a wide range of software, but the user experience can vary depending on the distribution you choose. It might be harder for non-technical users to navigate, which could be a drawback if your target audience isn’t familiar with Linux. | MacOS offers a smooth, user-friendly experience with a clean interface that’s easy for most users to pick up quickly. It’s especially strong in environments where users are already accustomed to Apple products. The downside is that developing and maintaining apps for MacOS can be more expensive due to the higher costs of hardware and software. It also requires developers with specific expertise in Apple’s ecosystem. | Windows is very popular and familiar to most users, making it a solid choice for a broad audience. It’s easy to find developers with experience in Windows environments, and the platform supports a wide variety of software. However, licensing costs are higher compared to Linux, and there are additional considerations for ensuring the app runs smoothly across different versions of Windows. |  iOS is known for its smooth performance and consistent user experience across devices. Developing for iOS can be advantageous because Apple users often expect high-quality apps, and the platform has a loyal user base. However, the development process can be more expensive and time-consuming due to strict guidelines and the need for specialized tools like Xcode.  Android offers a wide reach since it’s used on many different devices around the world. It allows for more flexibility in app design and features, but this comes with the challenge of ensuring the app works well across various devices with different screen sizes and hardware capabilities. The open nature of Android also means there are more potential security issues to consider. |
| **Development Tools** | On Linux, developers often use tools like Visual Studio Code, Atom, or Sublime Text, which are powerful and flexible. Linux also has strong support for command-line tools and package managers like , which make it easy to install and update software. While these tools are good, they may require more technical knowledge to use. | MacOS provides a polished development environment with tools like Xcode and Visual Studio Code, which are highly integrated with the system. These tools are user-friendly and offer great support for building apps, especially those targeting Apple devices. However, depending on specific Apple hardware and software can increase development costs, and developers need to be familiar with the MacOS ecosystem to make the most of these tools. | Windows development is supported by a wide range of tools, with Visual Studio being one of the most popular for creating Windows applications. The .NET framework and languages like C# are also widely used, providing a powerful set of features for developers. While these tools are comprehensive and well-documented, the cost of licenses and the need to maintain compatibility with multiple versions of Windows can be challenging. | Development for iOS typically uses Swift or Objective-C, with Xcode as the primary integrated development environment (IDE). Xcode offers powerful features like simulators and performance testing tools, making it easier to build and refine apps. However, the strict app review process and the need for a Mac to use Xcode can be limiting for some developers.  Android development is usually done using Java, with Android Studio as the main IDE. Android Studio is very big, offering emulators for testing across different device configurations. While Android’s open nature provides flexibility, it also requires more effort to ensure the app performs consistently across a wide variety of devices and operating system versions. |

## 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:  
To make it easy to scale "Draw It or Lose It" , I recommend using Linux as the primary operating platform. Linux is highly versatile, scalable, and cost-effective, making it ideal for both current and future needs. Its strong security features and extensive support for web and server applications will allow the game to be deployed across multiple devices, including desktops, tablets, and mobile phones, making sure that it reaches wide .**

**2-Operating Systems Architectures:  
I can suggest a architecture for Linux involves a client-server model with a multi-tier design. The client side will use a standard web technologies like HTML, CSS, and JavaScript to ensure compatibility across different browsers and devices. On the server side, the architecture will be divided into presentation, application, and data layers, allowing for better scalability, easier maintenance, and better security. This approach will also help future updates and feature expansions without causing problems to the user experience.**

**3-Storage Management:  
I am studying AWS and Linux so, I recommend using a combination of a relational database management system (RDBMS) like MySQL and AWS S3 for cloud storage. MySQL will efficiently handle structured data such as user profiles and game states, providing quick and reliable access. AWS S3 is ideal for storing large media assets like images and audio files, offering seamless scalability and accessibility from any location. This combination leverages my knowledge of AWS to ensure the game can manage increasing data volumes effectively while maintaining high performance.**

**4- Memory Management:  
With my background in Linux, I can say that Linux’s memory management is very reliable, using techniques like virtual memory and swap space to optimize available resources. modern web browsers running on client devices use garbage collection to automatically free up unused memory. This makes sure that "Draw It or Lose It" operates smoothly without manual intervention, minimizing the risk of memory leaks and ensuring responsive gameplay even under heavy usage. My studies in Linux have helped me with the knowledge to use these techniques, make sure the game’s stability and performance is a priority.**

**5- Distributed Systems and Networks:  
I can suggest the following, to help communicate between multiple platforms, I recommend using a distributed software architecture with servers hosted in the cloud. This setup will allow different devices to connect to a central server that manages game synchronization, real-time updates, and user interactions. To handle connectivity issues, such as network outages or low bandwidth, the system should include strong error handling and synchronization ways that ensure the game state remains consistent across all devices, even if some connections are temporarily disrupted.**

**6- Security:  
I think Security is a important, and the recommended Linux platform that helps with a big range of security features. I suggest using a safe communication protocols like HTTPS to safely encrypt data sent between client devices and the server. Also, using strong authentication methods such as two-factor authentication will help protect user accounts. Data stored in the cloud should be encrypted, with regular security checks and updates to address any vulnerabilities. This approach will ensure that user information is protected across all platforms.**