

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 | 07/20/2025 | Karon Bennett | Updated summary of The Gaming Rooms problems and goals. Also clarified what the requirements for this program will be as well as the constraints. |

**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)

Creative Technology Solutions is developing a game for The Gaming Room. The Gaming Room wants a web-based game called “Draw It or Lose It” that can be used on multiple platforms. The problem is that currently, this game can only be used on Android apps. The program also needs to meet all software requirements provided by the client. A solution to this problem would be to use a singleton pattern so that there is only one active instance in memory and to use cross-platform technology that can be accessed through multiple platforms.

## Requirements

Business Requirements:

* The game will be accessible through a web-based application so it can reach a larger audience rather than just Android users.
* Users must be able to create and join teams using unique names.
* The Gaming Room needs a streamlined development environment.

Technical Requirements:

* Singleton pattern so that there is only one instance of the game in memory at any given time.
* The game can have multiple teams.
* The game can have multiple players.
* Games and team names need to have unique identifiers.
* The game will have four rounds, each lasting one minute each.
* The image displayed in the game should slowly render, being fully revealed by 30 seconds.
* The other team has 15 seconds to guess the photo if the first team fails.
* The system needs to support loading and rendering from a large stock drawing library with timed visibility mechanics.
* The program must be built with web compatible software.

**Design Constraints**

The design constraints for developing the game application in a web-distributed environment are as follows:

* The game must be developed as a web-based application so that it can be used outside of Android devices.
* The game application must use the singleton pattern to ensure only one instance of the game is exists in the memory.
* The game must have four rounds per game, one minute per round, a 15 second guess opportunity for the other team if the first team fails to guess, and drawings completed at 30 seconds.
* The application must render stock images, no custom drawings.
* The environment must be simple, well documented, and easy to mention since The Gaming Room has limited technical knowledge.

## [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 UML class diagram shows how the different classes in The Gaming Room project are connected and how object-oriented programming (OOP) principles are used to meet the software requirements.

The classes are related in a few ways. The “SingletonTester” class depends on the “ProgramDriver” class, and “GameService” is directly connected to the Game class. The “Team” class is linked to both the “Game” and “Player” classes, which makes sense since a team is made up of players and is part of a game. The “Game”, “Team”, and “Player” classes all inherit from the “Entity” class, which holds common features like “id” and “name”.

The diagram demonstrates several OOP principles:

* **Encapsulation**: Most class variables are private (shown with a minus sign), and access to them is managed through getter and setter methods. This protects the data from being changed directly and allows for checks like making sure names are unique before adding them.
* **Abstraction**: The “Entity” class is an example of abstraction because it provides shared features that all the other classes can use. This helps avoid repeating code and makes the program easier to manage and update.
* **Inheritance**: The “Game”, “Team”, and “Player” classes inherit from the “Entity” class, which means they automatically get its fields and methods. This makes the code cleaner and easier to work with.
* **Polymorphism**: Since all the classes that inherit from the “Entity” class can override methods like “toString()”, each one can return its own version of the result. This means they can all be treated like an Entity but still behave differently, which helps keep the code flexible and maintainable.

**"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** | Mac can be used to host web apps, but it’s not common. It’s more expensive and not really built for large server use. It works well for testing or small projects but isn’t great for big websites. | Linux is the best choice for web servers. It’s fast, secure, free to use, and used by most companies. It can be hard to learn at first, but it’s reliable and works well for big projects. | Windows works well for servers that need Microsoft tools like .NET. It’s easy to manage but costs money and needs regular updates. It’s mostly used by big companies. | Phones and tablets can’t really host web apps. They don’t have enough power or battery life. They’re made to be users of apps, not hosts. |
| **Client Side** | Making apps for Macs takes more time and money because you need Apple computers and special tools. But if you use tools like React Native, you can make apps for multiple systems at once. | Not many people use Linux desktops, so it’s not usually a top priority. But it’s cheaper to develop for, and useful for software aimed at developers or tech users. Testing on different versions of Linux can take extra time. | Since Windows is the most used OS, you pretty much have to support it. It’s not too expensive, and there are good tools for building Windows apps. You may need to test on different Windows versions. | Mobile apps take a lot of time and skills, especially if you want to make separate apps for iPhones and Androids. Using tools like Flutter or React Native can help build for both at once and save time. |
| **Development Tools** | Max uses Xcode to build apps for Mac and IPhone. You can use other tools like VS Code or IntelliJ. It works well with many programming languages like Swift, Python, and JavaScript. | Linux works with tons of programming tools and languages like Python, Java, and PHP. It’s great for coding and testing web apps. You can customize it a lot, and it works well with tools like Docker. | Windows has tools like Visual Studio that are good for making apps, especially for .NET. It supports many languages like C#, JavaScript, and Python. It doesn’t work as well with some Linux tools unless you use WSL (Windows Subsystem for Linux). | Mobile development uses tools like Xcode (for iPhones) and Android Studio (for Android). You can also use React Native or Flutter to build apps for both. You’ll need to test apps on different phones and screen sizes. |

## 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**: I recommend using a cross-platform development framework like React Native or Unity, supported by a Linux-based server platform. This approach will let The Gaming Room expand “Draw It or Lose It” beyond Android to include iOS, Windows, Mac, and web browsers. React Native is ideal for mobile platforms, while Unity is perfect if the game has more graphics or real-time play. Linux is great for server-side hosting because it’s stable, secure, and free.
2. **Operating Systems Architectures**: The recommended architecture is a client-server model. The server (Linux-based) handles game logic, user accounts, and real-time data. The clients (on mobile and desktop platforms) connect through the internet to access the game. Each client device will run its own OS (like Android, iOS, Windows, or macOS) and communicate with the server over a network. By using a layered architecture, we can separate presentation, logic, and data, which makes it easier to update and scale.
3. **Storage Management**: For storage, cloud-based storage like AWS S3 or Google Cloud Storage is a good fit. These systems manage file storage efficiently and scale easily as more players join. Data such as user profiles, game history, and leaderboards can be stored in a relational database like PostgreSQL or MySQL, which work well with Linux servers. These storage options allow automatic backups and protect data from loss.
4. **Memory Management**: Linux uses virtual memory management to efficiently handle processes. It loads parts of programs into RAM only when needed (called “demand paging”), which helps with performance. The Linux kernel also uses caching to store frequently used data in memory, improving speed. For the Draw It or Lose It server, this means smoother gameplay and better handling of many users at once. On the client side, mobile devices also manage memory carefully to prevent apps from crashing or slowing down.
5. **Distributed Systems and Networks**: To allow Draw It or Lose It to work on different platforms and communicate in real time, we can use distributed systems. The game server handles central tasks, and the clients send and receive data over the internet using APIs or WebSockets. WebSockets are great for real-time communication (like showing drawing updates instantly). If one part of the system goes down (like the server), players may lose connection, so it’s important to use load balancers, redundancy, and failover systems to reduce downtime.
6. **Security**: Security is very important. All communication between the server and clients should be encrypted using HTTPS/TLS. User data like passwords should be stored using strong hashing algorithms (like bcrypt). We can also use authentication tokens (like JWTs) to keep sessions secure. Each platform (Android, iOS, Windows) has its own security features, and we should follow best practices for each. On the server side, firewalls, regular updates, and access controls will help protect against threats.