

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 | 05/25/2025 | Karen Schick | Defined executive summary, project requirements, design constraints, and domain model. |
| 2.0 | 06/08/2025 | Karen Schick | Defined the evaluation. |
| 3.0 | 06/22/2025 | Karen Schick | Defined recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has retained Creative Technology Solutions (CTS) to design a web-based version of Draw It or Lose It, currently only available on an Android app. The goal of this project is to expand the game’s reach across multiple platforms while sustaining its existing gameplay experience.

Draw It or Lose It is a competitive, team-oriented game based on the 1980s tv show Win, Lose, or Draw. During the game, a series of images are rendered, and a team attempts to guess what is drawn. If unable to guess correctly, other teams are given an opportunity to submit a guess. Each game consists of four one-minute rounds. The images are rendered at a steady rate and are fully complete within 30 seconds.

To meet The Gaming Room’s goals, the software must support games with multiple teams, allow each team to have multiple players, enforce unique game and team names, and ensure that only one instance of the game can exist at any time by creating unique identifiers for each instance of a game, team, or player. To meet these requirements CTS will develop an object-oriented solution with name validation and session management that ensures the game remain functional across all intended platforms.

## Requirements

* Web-based version
* Multiple platforms
* Games may have multiple teams
* Each team can have multiple players
* Game names must be unique
* Team names must be unique
* Only one instance of the game can exist at any time
* Each game, team, player must have unique id

## [Design Constraints](#_2et92p0)

* Web-based architecture: must have front-end and back-end components
* Cross-Platform compatibility: web-based and android
* Scalability: System must handle multiple users and teams at one time
* Uniqueness Enforcement: must validate that game and team names are unique
* Single Instance Enforcement: Only one instance of the game can exist at any time
* Identifier Management: Each game, team, player must have unique id

## [System Architecture View](#_ilbxbyevv6b6)

The system architecture for the web-based version of *Draw It or Lose It* is developed using a hybrid of client-server and serverless patterns. This enables cross-platform deployment, scalability, and cost efficiency. The application follows a three-tier architecture composed of the presentation tier (client), application logic tier (server), and data/services tier (cloud infrastructure).

## [Domain Model](#_8h2ehzxfam4o)

The Gaming Room UML class Diagram represents a domain model for the Draw It or Lose It game application.

At the top level, the ProgramDriver class contains the public main() method, which serves as the entry point of the application. It has a <<uses>> dependency on the SingletonTester class, which contains the public testSingleton() method used to validate singleton behavior.

The Entity class is the base class for Game, Team and Player. It defines encapsulated fields id and name with public getter methods. This demonstrates encapsulation and inheritance. Game, Team and Player all inherit the properties and behaviors from Entity.

The GameService class implements the singleton pattern through a private constructor and static getInstance() method. It manages a list of game objects, as well as counters for unique IDs related to games, players, and teams. The line marked “0...\*” between GameService and Game indicates a composition relationship as GameService is the only manager of Game instances.

The Game class consists of a list of Team objects and has similar composition relationship with Team. A Game can manage zero to many Teams through the addTeam() method.

The Team Class consists of a list of Player objects and can have zero to multiple Players. Like the other relationships it is also a composition as Players are created and managed within the context of a Team.

**"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 servers are known for their stability, reliability, and developer-friendly interface. Mac OS is Unix based, providing a large ecosystem of development tools. It supports standard web servers like Apache and Nginx and integrates well with open-source technologies. However, its high hardware costs make it less suitable for large-scale deployments making it less ideal for Draw It or Lose It. | Linux is the industry standard for deploying secure and scalable web applications such as Draw It or Lose It. Linux is highly customizable, free and open-sourced and supports all major web servers such as Apache and Nginx. This flexibility makes it an ideal fit for CTS to host the game. | Window is user-friendly and easily scalable. It has also great GUI- based management and commercial support and seamlessly integrates with other Microsoft services. However, historically, security was flawed, and licensing costs are high, making it a less attractive option for the Gaming Room when compared to Linux. | Mobile platforms are not designed for server-side hosting. While cost effective for small-scale use, they have performance and reliability issues, security concerns. Mobile devices are best suited to act as clients rather than web hosting. Draw It or Lose It should not be limited to solely client-side interaction if it wishes to reach a larger audience. |
| **Client Side** | On macOS, supporting Safari, Chrome, and Firefox ensures that *Draw It or Lose It* performs smoothly during image rendering and guessing rounds. However, expertise in Swift is required for native testing. Hidden costs include potential scaling, app store fees, and maintaining a robust testing infrastructure. Leveraging cross-platform frameworks like React or Flutter would reduce duplication of time and effort. | Linux desktop users typically rely on Chrome or Firefox. While licensing costs are low, because Draw It or Lose It is intended to be responsive and real-time, adapting software to different environments leads to lengthy development cycles. Expertise in Linux is required, and knowledge of cross-platform frameworks is helpful. | Windows supports a wide range of browsers, including Edge, Firefox, and Chrome. Extensive testing and development are needed to ensure compatibility across various Windows versions. Licensing costs can be significant. Thorough testing and API availability checks increase time. And developers must be proficient in Windows development tools such as C# and .NET. | Native development for IOS and Android offers the best performance for mobile devices, but they greatly increase development time and cost. Developers are required to be proficient in Swift for iOS or Java for android or use options such as React or Flutter for Cross-platform development. |
| **Development Tools** | Swift is Apple’s most intuitive language for native macOS apps. Other primary languages include C++, JavaScript, Python, Java and C#. Xcode is Apple’s official IDE. Visual Studio Code and JetBrains can also be used. Development is restricted to Apple hardware. For The Gaming Room, macOS is useful for testing client-side performance but not ideal for full-stack deployment due to hardware costs. | Linux supports a wide range of programming languages and open-source tools like VS Code, JetBrains, Eclipse and NetBeans. Java is the ideal language for object-oriented management in Draw It or Lose It. CTS can leverage Linux for both backend development and licensing costs. | Windows development tools like Visual Studio and VS Code support languages including C#, JavaScript, and Python. With Windows Subsystem for Linux (WSL), cross-platform development becomes easier. However, licensing for IDEs and Windows OS increases costs. Windows may be used for some testing or developer preference, but Linux is preferred for full deployment. | Native app development uses Swift/Xcode (iOS) and Kotlin/Java/Android Studio (Android). Cross-platform development can use React Native, Flutter, or Ionic. IDEs include VS Code, and Android Studio. For CTS, using a unified tech stack ensures maintainability across platforms and keeps costs down. |

## Recommendations

1. **Operating Platform**:

To support the expansion of Draw It or Lose It from an Android platform to a web-based, cross-platform application, I recommend using the Linux operating platform. Its open-source tools and engines as well as its compatibility with web servers like Apache and Nginx make it cost effective and highly scalable, making Linux an ideal fit for CTS to host web applications and manage back-end development.

1. **Operating Systems Architectures**:

A hybrid approach of combining serverless and client-server architecture will best support CTS’s expansion goals for Draw It or Lose It. Serverless architecture will handle storage and memory tasks such as asset storage and retrieval. While the client-server architectural pattern permits a separation of concerns. The client side is responsible for rendering the game’s user interface (UI) and handling user input, whereas the server side deals with maintaining the game’s core logic. This division of functions permits independent development enabling a single backend to support multiple front ends using a precisely defined API.

1. **Storage Management**:

Serverless architecture utilization optimizes cross-platform memory by off-loading image storage and retrieval operations to cloud services such as AWS lambda. It can be utilized to manage assets like game files and the image library, ensuring they persist between sessions. This minimizes the client-side effort and makes deployment across different platforms easier. Off-loading storage to the cloud allows for seamless cross-platform expansion of Draw It or Lose It.

1. **Memory Management**:

Effective memory management guarantees that Draw It or Lose It performs smoothly across all platforms, particularly given the game’s overall memory consumption of 1.6GB. Lazy loading methods save RAM by loading images only when they are required. Compression algorithms streamline image file sizes prior to rendering, conserving GPU and memory resources. Serverless Architecture provides storage and retrieval assets, including game files and the image library, to be accessed throughout sessions. Together, these techniques enable the game to render images every 30 seconds without exceeding mobile or browser memory capacities, providing consistent user experience.

1. **Distributed Systems and Networks**:

A distributed software architecture involves using the hybrid model above combining the client-server pattern and serverless architecture. The use of a RESTful API allows clients on various platforms to connect to the backend server over the network. The client sends HTTP requests to the server, which processes and returns the requested information. This communication between components ensures consistency across all platforms, even while recovering from network interruptions. The server is responsible for managing game logic and session state, while serverless components offload handle asset storage and retrieval. These event-triggered, stateless services increase reliability during system outages. To further enhance performance across techniques like lazy loading and compression improve network efficiency.

1. **Security**:

To protect user information across platforms, Draw It or Lose it will implement multiple layers of security. Role-based access control (RBAC) along with a centralized identity and access management (IAM) system will provide robust authorization management. RESTful server endpoints will be used to provide communication to the client from the server. An authenticator will verify correct username and password credentials, while an authorizer verifies the roles to determine whether the user is allowed to perform certain tasks. Frameworks, such as Dropwizard, can support secure interactions with these endpoints using annotations like @RolesAllowed. Furthermore, firewalls can control network vulnerabilities and antivirus software can add additional protection while regular system updates across all platforms are necessary as well to close known network vulnerabilities. Integrating these multi-level protections ensures application integrity and privacy of user data across various platforms.