

Draw It or Lose It – Web-Based Game Platform

**CS 230 Project Software Design Template**

Version 3.0

**Document Revision History**

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 08/16/25 | Aleksandr Donovan | Software evaluation of each platform’s advantages and limitations |

**Executive Summary**

Our client, the Gaming Room, currently operates a popular multiplayer Android game called Draw It or Lose It. To expand their operations, they want to develop a web-based version that supports multiple platforms. This new web-based application will retain the core gameplay of teams guessing rendered drawings within timed rounds. But we will replace the traditional method of players drawing on an easel with pre-rendered stock images. Additionally, the updated application will introduce improved user features, gameplay synchronization, and secure cross-platform functionality.

To achieve this, the proposed solution involves creating an object-oriented design that incorporates proven software design patterns, such as Singleton and Iterator. These patterns ensure that only one game instance runs at the same time, and that team and game names remain unique. By developing this scalability, The Gaming Room will be able to broaden its audience and position themselves for sustainable growth.

**Requirements**

* *Develop a web-based version of Draw It or Lose It supporting multiple platforms*
* *Implement pre-rendered stock images rather than use player-drawn images for the game.*
* *Ensure game and team names are unique.*
* *Restrict the system to a single active game instance at any given time. (Singleton pattern)*
* *Support real-time gameplay synchronization across all connected platforms.*
* *Implement improved user-management functionality.*
* *Incorporate secure cross-platform communication and protect user data.*

**Design Constraints**

* **Web-Based Accessibility:** The game must function consistently across various platforms and web browsers.
* **Environment:** Since the application will support real-time multiplayer gameplay across distributed devices, it will require efficient network communication.
* **Singleton Implementation:** Only a single instance of the game can exist at any given time. This means it is necessary to implement Singleton design patterns within the game's architecture to manage resources and ensure application state consistency.
* **Unique Naming Constraints:** The app requires unique team names. This means validation checks during creation and registration processes. This constraint impacts the database which must efficiently manage these validations.
* **Security and Privacy:** User data must be protected. Secure communication and encryption methods must be used to meet data privacy and compliance standards.
* **Extensible Object-Oriented Design:** The application's architecture must support scalability and easy maintenance. Using object-oriented design principles will help future expansions without re-engineering the application.

**System Architecture View**

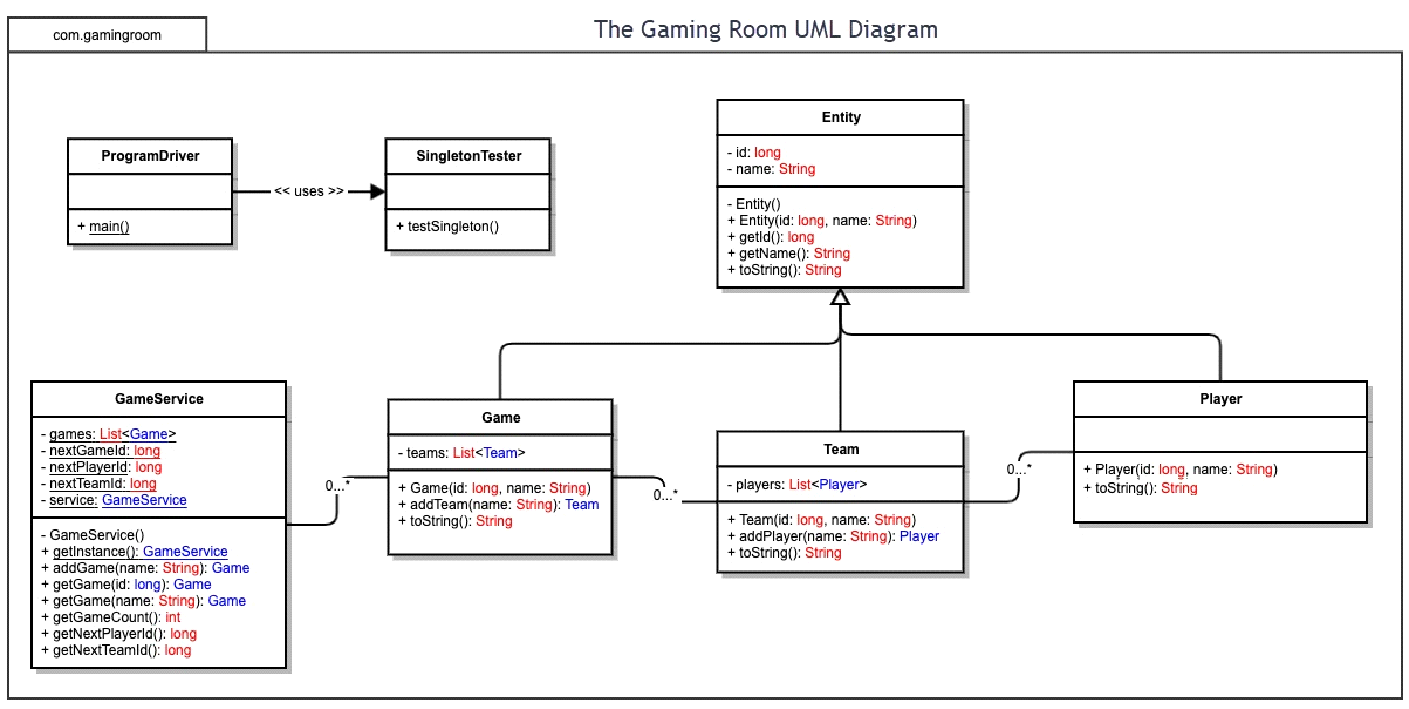
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

**Domain Model**

The UML class diagram for Draw It or Lose object-oriented design has four primary classes: Game, Team, Player, and Entity. At the heart of this model is the Entity class. This class contains two attributes: id and name. These attributes are inherited by all other classes for reusability and to reduce redundancy in the code.

The Game class contains a list of Team objects, establishing one-to-many relationships where a single game instance can include multiple teams. Similarly, the Team class holds a list of Player objects to allow each team to have multiple players. These relationships are represented using aggregation notation in the diagram.

The GameService class manages the overall application logic. It uses the Singleton pattern to ensure that only one instance of the service exists in memory at any given time.  Additionally, it handles ID tracking for games, teams, and players while providing methods for adding and retrieving game instances.



**Evaluation**

* Evaluation of various platforms for their characteristics, advantages, and weaknesses for hosting a web-based software application.
* Determining the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients.
* Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on each operating platform.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS can host web applications through servers like Apache or Nginx, which can run modern web stacks. And tools like Homebrew allow for the installation of server software. However, macOS is rarely used in production web hosting environments due to higher hardware and initial licensing costs. As well as limited server support, and a smaller hosting ecosystem. While useful for development and testing, macOS is not ideal for scalable public-facing server deployment. It is typically used for local development rather than public deployment, like the Draw It or Lose It app. | Linux is a widely used industry standard platform for hosting web-based applications. Being open-source, it supports common applications such as Apache, Nginx, and Node.js, as well as containerization tools like Docker and Kubernetes.  Linux's open-source nature also eliminates licensing fees that are common in macOS. The stability and security make it ideal for distributed, scalable environments, such as cloud hosting on AWS, Azure, or Google Cloud. It would be the best platform for hosting Draw It or Lose It at scale. | Windows Server uses Internet Information Services (IIS) for hosting web applications. It is also compatible with Microsoft frameworks, such as .NET, while being reliable and offering good administrative tools.  The downside is that it also uses paid licensing and has fewer open-source tools compared to Linux. So while hosting Draw It or Lose It is possible, it would be costlier and more resource-intensive than Linux-based solutions. | Android and iOS cannot host web applications due to hardware limitations, power constraints, and security risks.  Mobile devices are often just used as clients in a distribution system. However, they are essential to consider for frontend compatibility and must connect to the centralized web application hosted on a server platform such as Linux. |
| **Client Side** | Developing for macOS would once again provide some complications. Not only do you have to provide compatibility with the Safari browser, which behaves differently from other browsers, but Apple’s development systems tend to be more closed off compared to other operating systems. You also require Macs for actual testing and development.  This will increase the cost and limit the flexibility of development. However, because macOS users might adopt the latest system updates more often, the game will benefit from consistent browser versions. | Linux desktop users often use Firefox or Chromium-based browsers. And while Linux is not as popular as Windows or Mac, it adheres closely to open web standards, making compatibility easier.  Fortunately, testing on Linux is cost-effective and can be done easily using virtual machines or containers compared to Mac or Windows.  That is why Developers should ensure their application runs reliably across major Linux distributions, since it would be perfect for hosting the game as well. | Windows is the most widely used desktop operating system, so ensuring compatibility is essential. This means that Developers need to test across multiple browsers and system configurations.  Fortunately, development for Windows is straightforward and cost-effective as it is well supported by web development tools.  Many developers are also already familiar with Windows environments, given how popular it is. | Supporting mobile platforms adds additional complexity to the development. Developers must create a responsive design that adapts to different screen sizes, resolutions, and orientations.  Expertise in mobile input and performance optimization is required before testing on both Android and iOS.  Time and cost will increase due to different developer fees and mandatory hardware. Especially on the iOS side of things, as their requirements are more strict and costly. |
| **Development Tools** | Mac supports a wide range of web development tools. This includes Visual Studio Code, IntelliJ IDEA, and Eclipse. Unfortunately for iOS-specific testing or development, Xcode is required and it only runs on macOS. This means that Mac is essential for supporting and testing the Draw it or Lose it mobile version in Safari. Requiring Mac hardware may limit flexibility and increase cost for the team. | Linux would be a strong platform for both backend and front-end development. Tools like Node.js, VS Code, Docker, and Git are all supported on the platform. Linux is also ideal for deploying the game servers. Especially when using containerized services or WebSocket-based real-time communication.  And since Draw It or Lose It requires fast processing, like image rendering and timer tracking, the support for server-grade tools makes it the best option for development and deployment. | Windows will support all major development tools that would be required for Draw It or Lose It.  It is commonly used for front-end work and offers good debugging tools.  The down side is that it is not optimized for Linux-native server tools, but with Windows being accessible and familiar for many developers, it is a solid environment for building and testing the game client. Developers can also test in Microsoft Edge and Chrome to ensure browser compatibility. | The development for mobile will be focused on ensuring performance in mobile browsers rather than building native apps.  The Gaming Room can use tools such as Chrome DevTools for mobile, Appium, or cloud services like BrowserStack to verify that Draw It or Lose It performs as intended. Cross-platform frameworks like React Native can also help the team with UI for all platforms. This will help reduce the need for separate front-end/mobile teams. |

**Recommendations**

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**:

For hosting the web-based version of Draw It or Lose It, Linux would be the best recommendation for an operating platform. Specifically, server-grade distributions such as Ubuntu Server or CentOS deployed on a cloud environment. They are both popular and are well-suited for server environments for this game.

Linux is the industry standard for large-scale web application hosting as it is open-sourced, stable, secure, and highly customizable with no licensing fees, unlike Mac. This supports the project’s requirement for scalability and cost efficiency while also ensuring uptime for multiplayer sessions.

Modern web stacks such as Node.js, Nginx, and Apache are also compatible with Linux. And container tools like Docker or Kubernetes allow for efficient resource allocation. This will support the game on all major desktops and mobile platforms via modern web browsers.

* **Operating Systems Architectures**:

The recommended platform architecture for hosting *Draw It or Lose It* on Linux follows a multitiered design where you separate content delivery, application logic, and data storage. This separation provides scalability and easier maintenance while allowing access from Windows, Mac, Linux, and mobile clients.

Content delivery networks will deliver static assets such as stock drawings, JavaScript, and CSS files from servers close to players to reduce latency and help minimize load times. Incoming requests will pass through a load balancer and reverse proxy, which will both handle encryption for secure connections and route the traffic to the correct application service. This means that the app will run inside a container using Kubernetes or something similar to ensure that game APIs can scale to meet player demand.

The data will consist of a relational database to store persistent information like user profiles, team records, and game history to ensure quick reads for real-time gameplay and keep permanent records. Linux is once again a good platform for this with built-in security controls, making it well-suited for running this type of game.

This architecture will keep the gameplay synchronized across all connected clients, and any network interruptions or server failures are handled gracefully through automatic reconnection and state restoration.

* **Storage Management**:

For the game, the storage should be organized to efficiently handle different types of data. A relational database will store permanent records such as user profiles, while large static assets like drawings will be stored in cloud-based storage to be quickly provided to players during gameplay. This will allow for more frequently accessed files to be delivered from servers closer to the players.

Regular backups will be set up for both the database and object storage to recover previous states if needed. This approach keeps the game responsive, protects important data, and supports future scalability.

* **Memory Management**:

A Linux-based server offers advanced memory management features that help maintain consistent performance during high player activity. Linux utilizes virtual memory and paging to ensure that only the necessary data resides in memory, thereby minimizing resource usage when multiple players are connected to the game.

To improve performance, the server will use in-memory caching systems like Redis to store active game data. This reduces the need for repeated database queries each round. In addition, the Linux process scheduler balances CPU and memory usage across the game’s containers. Together, this ensures that as more players join, the game maintains smooth experience as the number of players increases.

* **Distributed Systems and Networks**:

For real-time multiplayer gameplay across various platforms, Draw it or Lose it should be built using a distributed architecture. This will allow clients and servers to communicate efficiently over the network during the games. WebSocket would be the most suitable technology for this kind of multiplayer game.

WebSocket enables a persistent, full-duplex connection between the client and server. This low-latency, two-way communication ensures that updates such as drawings, scores, and timers are delivered instantly, making it ideal for applications requiring instant updates.

The architecture will also need to be designed to handle common network challenges. Such as intermittent connectivity or server restarts. The application should automatically reconnect when needed and restore the session so the players can rejoin a game without losing any progress.

Additionally, distributed components like game servers and user interfaces must be synchronized. Even across different operating systems and devices. To ensure consistent gameplay, we need to account for server outages and connection dropouts so that the user experience remains smooth and responsive to different network conditions.

* **Security**:

To secure user data, all communication between clients and the server should be encrypted using HTTPS. Sensitive information will be hashed before storage to ensure it cannot be recovered in plain text if the database is compromised. Input validation will also be implemented on both client and server sides to block any attempted injection attacks or malicious requests.

Access to server resources should be restricted through role-based permission. This will limit certain actions to authorized personnel only. Together, these measures will protect sensitive data and maintain the game's security across all supported platforms.