

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 |
| --- | --- | --- | --- |
| 2.0 | 06/7/25 | Deonte Tate | Conducting an evaluation of the characteristics, advantages, and weaknesses of each platform (Linux, Mac, and Windows), as well as mobile platforms, for how the game application software could be deployed and run and what would be required to do so. |

**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)

Draw It or Lose It, is a web-based multiplayer game, where teams compete to guess a puzzle based on a series of progressively revealed stock images. The application will consist of timed rounds, each lasting one minute, with images becoming fully visible at the 30-second mark. If the team fails to solve the puzzle within the time limit, other teams will have a brief opportunity to make a single guess. This setup requires a streamlined and scalable software design to support multiple teams, timed gameplay, and real-time user interaction.

Each game will be represented by a unique instance that can only exist once in memory at any given time. Unique identifiers will be used to manage game, team, and player instances, ensuring no duplication. This design will also allow players to check the availability of names before selecting them.

## Requirements

Business Requirements:

1. **Multiplayer support:** The game must allow for one or more teams to participate in each game session.
2. **Team Composition:** Each team must be able to include multiple players.
3. **Unique Naming:** Game names and team names must be unique, allowing users to check the availability of a name when joining or creating a team.
4. **Single Instance:** Only one instance of the game should exist in memory at a time to prevent duplication and ensure smooth gameplay management.

Technical Requirements:

1. **Unique Identifiers:** Each game, team, and player must have a unique identifier for tracking and management purposes.
2. **Image Rendering Timeline:** Clue images must be fully displaced by the 30-sec mark in a 60 second round.
3. **Timed Gameplay:** Each round will last one minute, with a 15-second guessing window for other teams if the primary team fails to solve the puzzle.
4. **Web-Based Platform:** The game will be developed as a web application to ensure accessibility across platforms and devices.
5. **Real-Time Interaction**: The application must support real-time communication and gameplay updates for all participating users.

## [Design Constraints](#_2et92p0)

1. **Web-based environment -** This constraint requires the application to be platform-independent, responsive, and optimized for varying network conditions and devices. This means it will rely on languages such as HTML, CSS, JavaScript, as well as backend support frameworks. Developers must ensure cross-browser compatibility and implement responsive design principles for smooth user experiences on both desktop and mobile devices.

**Implications:**

* Introduces complexity in maintaining application state across multiple clients and sessions.
* Limits the use of device-specific features unless managed carefully through web APIs.
* Requires robust client-server communication to support real-time gameplay and updates.

1. **Real-Time-Multiplayer Interaction** - The game must allow real-time interaction among teams and players during gameplay. This requires fast and consistent data synchronization across multiple users.

**Implications:**

* Use of real-time technologies like WebSockets to minimize latency and deliver immediate updates.
* Error handling and reconnection logic must be built in to handle intermittent connectivity issues.
* Increases server load and complexity, requiring efficient event handling and state management on the backend.

1. **Single Game Instance in Memory -** Only one active instance of a game should be stored in memory at a time. This constraint aims to reduce redundancy and ensure consistent gameplay management.

**Implications:**

* Requires centralized session and state management using tools like in-memory data stores or singleton patterns on the backend.

1. **Unique Naming** - Game and team names must be unique and verifiable during creation or joining.

**Implications:**

* Requires backend logic for validating and storing names in real-time to prevent conflicts.
* Must handle race conditions where two users attempt to register the same name simultaneously.

1. **Timed and Progressive Rendering of Images -** Clue images must render gradually and be fully visible by the 30-second mark.

**Implications:**

* Client-side rendering logic must include timed intervals for progressive image display.
* Requires preloaded image assets and reliable timing mechanisms to ensure smooth transitions regardless of device performance or latency.
* Time synchronization between client and server must be accurate to ensure fair gameplay.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The UML class diagram for the Draw It or Lose It application represents the entities involved in the game system and how they interact with each other. The **Entity** super class is at the center of the model, defining shared attributes, id and name. Two classes, **Team** and **Player** inherit from the Entity class, demonstrating the OOP principle of inheritance. This promotes code reusability by allowing shared functionality to be defined only once in Entity and used in all derived classes.

The **Team** class contains a collection of Player objects, which is an example of a composition relationship ("has-a"). This means each team is composed of one or more players, supporting the requirement that "each team will have multiple players assigned to it." The use of a list or collection within Team allows for flexible team management and aligns with the game’s multiplayer structure.

Encapsulation is also demonstrated in this domain model, as each class manages its own data. For example, a player managing their team ID, or a team managing its players. By defining unique ids for each Entity, the model supports the requirement for uniquely identifiable game components, ensuring name uniqueness and enforcing the rule that only one game instance can exist in memory.

Together, these object-oriented principles—inheritance, encapsulation, and composition—ensure a scalable and maintainable design that fulfills the client’s functional and technical requirements efficiently.

**"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** | macOS can host web-based applications using tools like Apache or node.js.  However, **macOS is not licensed for use as a server hosting platform**, making it unsuitable for scalable or production server deployment.  Its Unix-based system is a strength for development but may be impractical for scalable deployment.  It is not commonly used in production environments due to limited server support and higher hardware costs. | Most widely used platform for hosting web applications.  It is open-source, stable, highly secure, and supports a wide range of server-side technologies.  offering excellent performance, low cost, and community support. | Windows Server is robust and integrates well with .NET technologies  It has higher licensing costs and is less favored for deploying JavaScript or Python-based web applications.  It's better suited for using Microsoft technologies. not be the most cost-effective option. | Mobile devices are not used for hosting server applications due to hardware limitations, power constraints, and lack of persistent connections. |
| **Client Side** | Requires consideration of Apple’s specific UI guidelines and system behavior.  Cost and development time can be moderate  Cross-platform testing is essential to ensure compatibility. | Client-side development considerations include testing on browsers like Firefox and Chromium.  Cost is low, but developer expertise in Linux GUI behavior may be necessary.  Compatibility testing is crucial but less intensive due to standards-compliant browsers. | Most widely used desktop OS, so it's critical to ensure compatibility.  Testing on Microsoft Edge and Chrome is important.  Development cost and time are average, but there is a strong developer community and good tooling support. | Mobile client support is essential, as users will likely access the game from phones or tablets.  Requires responsive design and testing across iOS and Android browsers.  Significantly adds to development time and cost and requires expertise in mobile-first design and touch-based UI interactions. |
| **Development Tools** | Supports tools like Xcode, VS Code, and JetBrains IDEs for developing web applications.  Also supports backend frameworks such as Node.js, Django, and Spring Boot.  Useful when iOS testing is required, as it’s the only platform supporting Xcode. | Offers robust command-line tools, package managers (apt, yum), and support for IDEs like VS Code, Eclipse, and IntelliJ.  Ideal for backend development using open-source stacks such as MEAN, LAMP, or Django.  It's highly efficient and customizable for developers. | Supports a wide variety of development tools including Visual Studio, VS Code, and IntelliJ.  It works well with backend stacks like ASP.NET, Node.js, and Python.  Also supports Docker and WSL2, improving compatibility with Linux-based tools. | Requires platforms like Android Studio (for Android) and Xcode (for iOS, only on macOS).  Tools such as React Native or Flutter can help with cross-platform development.  Testing tools like BrowserStack or physical devices are needed for accurate results. |

## 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**:

A Linux-based server platform is the most appropriate choice for expanding Draw It or Lose It across various computing environments. Linux is renowned for its stability, scalability, and high security—core requirements for a multiplayer, cross-platform game. Its open-source nature minimizes licensing costs, making it budget-friendly for scaling operations. Furthermore, Linux supports modern development stacks, containerization tools like Docker, and real-time communication technologies necessary for hosting concurrent multiplayer sessions. These characteristics make Linux ideal for both development and production deployments in local data centers or cloud environments.

1. **Operating Systems Architectures**:

Linux operates on a monolithic kernel architecture, where core services such as device drivers, memory management, and file systems run in a unified address space. This design results in better performance and easier communication between subsystems. Linux efficiently supports 64-bit architectures, which are standard in modern server hardware, ensuring robust performance and compatibility.

In addition, Linux’s modularity allows custom kernel builds and fine-grained control over system resources—critical when optimizing for performance in high-demand multiplayer environments. Tools like Docker and Kubernetes enhance this flexibility by enabling containerized deployments across heterogeneous environments, ensuring consistency and efficiency.

1. **Storage Management**:

For scalable and flexible storage on Linux, the Logical Volume Manager (LVM) is recommended. LVM allows dynamic resizing of partitions and simplifies the addition or removal of storage volumes without downtime—ideal for a game platform expecting growth in user data, session logs, and content updates. LVM also supports snapshots and backups, providing versioned states of the storage environment that can be restored in case of failure or corruption, helping maintain data integrity and continuity of service.

1. **Memory Management**:

Linux implements several advanced memory management techniques, making it well-suited for applications like Draw It or Lose It:

Virtual Memory: Uses disk space to simulate additional RAM when physical memory is full.

Paging and Swapping: Moves inactive memory pages to swap space to free up RAM for active processes.

Memory Isolation: Ensures that each running process has a protected memory space, enhancing both stability and security.

Shared Libraries: Reduces memory footprint by allowing multiple processes to use common code in memory.

These capabilities help optimize performance and ensure consistent user experiences, even during peak loads.

1. **Distributed Systems and Networks**:

To support cross-platform functionality, *Draw It or Lose It* should be designed as a **distributed application**. The game can be deployed across multiple server instances using a **microservices architecture**, where components (e.g., matchmaking, user authentication, game sessions) are decoupled and communicate over the network via APIs.

**Key considerations include:**

* **Scalability**: Horizontal scaling via cloud-based services or container orchestration (e.g., Kubernetes).
* **Fault Tolerance**: Redundant services and load balancers ensure continued operation during hardware or software failures.
* **Interoperability**: RESTful APIs or WebSockets enable communication between clients on different platforms (e.g., iOS, Android, Windows).
* **Latency Management**: CDN integration and regional server deployment minimize lag and improve responsiveness.

These distributed network principles ensure reliable and consistent gameplay across devices.

1. **Security**:

Security is paramount when handling user data across platforms. A Linux-based system can implement comprehensive, multi-layered protection for *Draw It or Lose It*:

1. **In Transit**: Data should be encrypted using **TLS/SSL (HTTPS)** to prevent eavesdropping or tampering.
2. **At Rest**: Files and databases should be secured with encryption tools like **LUKS** or built-in Linux file permissions.
3. **Access Control**: Role-based access and authentication protocols (OAuth, multi-factor authentication) should restrict sensitive operations.
4. **System Hardening**: Regular updates, firewalls (iptables/ufw), and intrusion detection systems (e.g., Fail2Ban) protect against vulnerabilities.
5. **Audit Logging**: Tracks user actions and system access to aid in security monitoring and incident response.

Linux’s built-in tools, combined with third-party security solutions, offer a strong defense against threats while ensuring compliance with data privacy standards.