

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 4**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  2.0  3.0 | 05/26/2024  06/09/2024  06/23/2024 | Corey Peterson  Corey Peterson  Corey Peterson | Software Design Document Original Upload  Instructor Feedback Applied  Recommendation Section Added |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based multiplayer game called "Draw It or Lose It," inspired by the classic game show "Win, Lose or Draw." This engaging game will allow teams of players to compete in guessing phrases or titles based on progressively revealed stock drawings. The game will be designed for web browsers, ensuring accessibility across various devices and platforms.

A key challenge is creating a real-time, interactive experience for multiple teams playing simultaneously. This will involve managing the game state, synchronizing drawing reveals, handling user input, and preventing cheating. The nature of web-based systems demands robust error handling and security to protect user data and ensure fair play.

Our proposed solution utilizes modern web technologies and a client-server architecture. The server will handle game logic, image rendering, and client communication. Clients will be responsible for rendering and processing the user interface and input. A database will store game data, user information, and historical records.

We will adopt an agile methodology with regular feedback loops and iterative improvements to ensure a smooth development process. This will allow us to adapt to evolving requirements and deliver a high-quality product that meets The Gaming Room's expectations.

## Requirements

* Multiple teams with multiple players per team
* Unique game and team names
* Real-time drawing reveals synchronized across all clients
* Turn-based guessing with time limits
* Secure handling of user data
* Cross-platform compatibility (web browsers)

## [Design Constraints](#_2et92p0)

* **Web-based Environment:** The game must operate within the constraints of web browsers, including limitations in processing power, memory, and network latency. This may necessitate optimizing image rendering, minimizing data transfer, and employing efficient algorithms.
* **Distributed Nature:** Handling multiple simultaneous games and ensuring a consistent state across all clients requires careful synchronization and conflict resolution mechanisms. We must address network latency, connection disruptions, and potential cheating attempts.
* **Security:** Protecting user data and preventing cheating is paramount. We will implement robust authentication, encryption, and input validation measures to ensure a secure and fair gaming experience.
* **Scalability:** The design should allow for future expansion, potentially accommodating many concurrent players and games. This may involve load balancing, database optimization, and efficient server-side processing.

## [System Architecture View](#_ilbxbyevv6b6)

N/A

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram represents the core entities and relationships within the Draw It or Lose It game:

* **Game:** The central object representing a single game instance. It holds information about teams, the current round, the puzzle, and the game state.
* **Team:** This represents a group of players participating in a game. It stores the team's name, score, and list of players.
* **Player:** This represents an individual user participating in a game. It stores the player's name, team affiliation, and potentially other relevant data.
* **Puzzle:** Represents the phrase or title to be guessed. It stores the text of the puzzle and possibly additional information like hints or categories.
* **Drawing:** Represents the visual clues provided to the players. It could store image data or references to image files.

The diagram demonstrates several object-oriented principles:

* **Encapsulation:** Each class encapsulates data and behavior, promoting modularity and code reusability.
* **Association:** Classes are linked through associations (e.g., a Game has Teams, a Team has Players), representing the relationships between objects.
* **Aggregation:** The Team class aggregates Players, indicating a "has-a" relationship where a Team comprises multiple Players.

**"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** | Macs, particularly with macOS Server, can host web applications. However, they are less common in server environments due to higher costs and less extensive community support than Linux. | Linux is the dominant choice for web servers due to its stability, flexibility, open-source nature, and vast community support. It offers various web server software options (Apache, Nginx) and robust tools for server management. | Windows servers can also host web applications using IIS. However, they are less common in the open-source community and might require additional licensing costs. | Due to limited processing power, storage, and bandwidth, mobile devices are unsuitable for hosting web-based software applications. They are primarily client devices for accessing applications hosted on servers. |
| **Client Side** | Developing for Mac clients is straightforward, as most web technologies are well-supported. However, the relatively minor market share compared to Windows might require additional testing and optimization. | Developing for Linux clients is similar to Mac, with good support for web technologies. However, the diverse range of Linux distributions might require broader testing to ensure compatibility. | Windows is the most widely used desktop operating system, making it crucial to ensure compatibility for Windows clients. Most web technologies are well-supported, but thorough testing is necessary due to the diverse versions and configurations. | Mobile devices present unique challenges due to varying screen sizes, touch input, and performance limitations. Responsive web design and mobile-specific optimizations are necessary for a good user experience across different devices and operating systems (iOS, Android). |
| **Development Tools** | Xcode is the primary IDE for Swift and Objective-C development on Macs. Other popular tools include Visual Studio Code, Sublime Text, and web development frameworks like React, Angular, or Vue.js. | Linux offers various development tools, including text editors (Vim, Emacs), IDEs (Eclipse, IntelliJ IDEA), and command-line tools. The choice often depends on personal preference and project requirements. Web development frameworks are also well-supported on Linux. | Windows developers often use Visual Studio, a powerful IDE with extensive features for various programming languages. Other popular choices include Visual Studio Code, Sublime Text, and web development frameworks. | Mobile development involves platform-specific tools like Xcode (iOS) and Android Studio (Android). Cross-platform frameworks like React Native or Flutter can be used to build applications that work on iOS and Android, but they might have limitations compared to native development. |

## Recommendations

Based on the evaluation of different platforms and considering the requirements for "Draw It or Lose It," here are my recommendations:

1. **Operating Platform:**
   * **Recommendation:** Linux
   * **Reasoning:** Linux is the ideal choice for hosting the "Draw It or Lose It" game server due to its:
     + **Cost-effectiveness:** It is open-source and free to use, reducing the overall project cost.
     + **Stability and Performance:** Linux servers are known for their reliability and efficient resource utilization, crucial for handling real-time multiplayer games.
     + **Community Support:** The vast Linux community ensures extensive documentation, troubleshooting resources, and ongoing development of server software.
     + **Flexibility and Customization:** Linux allows for tailoring the server environment to the game's specific needs, optimizing performance and resource allocation.
2. **Operating Systems Architectures:**
   * Linux servers can be deployed on various architectures, including x86-64 (most common), ARM, and others. The choice depends on the specific hardware being used.
   * For the client side, the game will be web-based and thus compatible with most modern operating systems (Windows, macOS, Linux) and mobile devices (iOS, Android) through web browsers.
3. **Storage Management:**
   * For storing game data, user information, and historical records, a relational database management system (RDBMS) like MySQL or PostgreSQL is recommended. These provide robust data storage, indexing, and querying capabilities, essential for managing game state and user data.
   * For storing image files (drawings), a scalable object storage system like Amazon S3 or similar cloud storage services can be utilized. This allows for efficient storage and retrieval of large volumes of images.
4. **Memory Management:**
   * Linux employs virtual memory management, allowing each process to have its own address space and protecting it from other processes.
   * It also utilizes memory management techniques like demand paging, swapping, and caching to optimize memory usage and performance.
   * For a real-time game like "Draw It or Lose It," monitoring memory usage and optimizing the game logic and data structures to avoid memory leaks and ensure smooth gameplay is crucial.
5. **Distributed Systems and Networks:**
   * To facilitate communication between the server and multiple clients, a client-server architecture using the WebSocket protocol is recommended. This enables real-time, bidirectional communication, which is essential for synchronizing game state, sending drawing updates, and handling user input.
   * To handle many concurrent players, load-balancing techniques can be employed to distribute the workload across multiple servers.
   * Network connectivity, outages, and latency issues should be considered and addressed through robust error handling and reconnection mechanisms.
6. **Security:**
   * Implement secure authentication mechanisms (e.g., username/password, OAuth) to protect user accounts and prevent unauthorized access.
   * Use HTTPS to encrypt communication between the client and server, safeguarding user data from eavesdropping.
   * Input validation and sanitization are crucial to prevent injection attacks and other security vulnerabilities.
   * Regularly update server software and libraries to patch any security vulnerabilities that may be discovered.