

# 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 5**](#_Toc115077325)

[**Recommendations 9**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 02/25/2024 | Michael Crevier | The overview of a web-based, multiplayer game known as Draw-It Or Lose-It. Recommendations section was revised to include more detailed explanations on cross-platform deployment, distributed systems handling, and specific security measures, providing stakeholders with clearer guidance on implementing a robust, scalable, and secure gaming platform across various environments. |

**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 (CTS) is thrilled to collaborate with The Gaming Room on the development of a cutting-edge web-based version for their Draw It or Lose It game. Our software design will leverage modern technologies, including robust frameworks compatible with diverse operating systems such as Windows, macOS, iOS, and Linux. The application will be developed using responsive web design principles, ensuring seamless gameplay across various platforms and devices. Key features include support for multiple teams each featuring multiple players, unique naming conventions for exclusivity of team and player names, and unique identifiers for each game instance, allowing for only one game to exist in memory at a time. CTS is committed to delivering a comprehensive software design document and an implementation plan that aligns with The Gaming Room's vision.

## Requirements

From a technical perspective, the client requires seamless browser compatibility (Chrome, Firefox, Safari, Edge), support for operating systems such as Windows, macOS, and Linux, robust security measures, scalability to manage varying user loads, and cross-device functionality. On the business side, the primary focus is streamlining the extension of their Android game, Draw It or Lose It, into a multi-platform web-based application. Emphasis is placed on collaborative team play, accompanied by the implementation of unique naming systems for both games and teams, all geared towards enhancing user engagement.

## [Design Constraints](#_2et92p0)

The game will need to be compatible with a wide range of platforms and browsers with distinct standards and features. Compatibility testing will be necessary and must be both extensive and done with frequency. The application must be scalable to deal with the varied performance loads and aspect ratios of the different types of users and their systems. As a web-based application, network latency may create delays which must be anticipated and dealt with in order to maintain a good user experience. Likewise, web-based applications present a range of security challenges which must be wrestled with, including encryption, input validation and authentication. This will require consistent updates and audits to identify and mitigate external threats. The application will need to comply with web standards, including HTML and Javascript standards to ensure universal usability. The graphical user interface must be responsive and dynamic in order to adapt to the various devices that will be used to play the game. Data will need to be synchronized in real time across multiple users in order to maintain the ability to play the game in multi-player fashion.

## [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 for the game application provides a birds-eye view of the relationship between the various classes of the program, showing how they behave and interact with each other.

1. Entity: This base class sets out the common attributes and methods utilized by the other classes in the program. The variables it sets out are ‘id’, corresponding to the numeric game ID (of type long) and ‘name’, corresponding to the text name of the game (of type String). The methods that are utilized are:
   * Entity(), a default constructor
   * Entity(id, name), a constructor which accepts the id and name parameters
   * getId(), a getter for the id variable
   * getName(), a getter for the name variable
   * toString(), a method which creates an output statement joining the id and name variables
2. Game: A class that instantiates an individual game session, extending Entity to use the methods above. The methods utilized are:
   * Teams, an ArrayList holding the list of active teams for the game.
   * Game(id, name), a constructor for the game that takes parameters id and name.
   * addTeam(name), a method to add a team to the current game using the name parameter
   * toString(), a method which creates an output statement joining the id and name variables
3. Team: A class that instantiates an individual team, extending Entity to use the methods above. The methods utilized are:
   * players, an ArrayList holding the list of active players for the team.
   * Team(id, name), a constructor for the team that takes parameters id and name.
   * addPlayer(name), a method to add a player to the current team using the name parameter
   * toString(), a method which creates an output statement joining the id and name variables
4. Player: A class that instantiates an individual player, extending Entity to use the methods above. The methods utilized are:
   * Player(id, name), a constructor for the game that takes parameters id and name.
   * toString(), a method which creates an output statement joining the id and name variables
5. GameService: The GameService class serves as a central component for managing game instances within the program. Employing a singleton pattern, it ensures a singular instance with static variables tracking active games, player identifiers, and team identifiers.
   * addGame(name), Constructs and returns a new or existing game instance based on the provided unique name.
   * getGame(id), Retrieves the game instance with the specified ID.
   * getGame(name), Retrieves the game instance with the specified name.
   * getGameCount(), Provides the count of currently active games.
   * getNextPlayerId(), Returns the next player identifier.
   * getNextTeamId(), Returns the next team identifier.
6. ProgramDriver: The ProgramDriver class serves as the entry point for the program, containing the main method. It initializes a singleton instance of the GameService class, ensuring only one instance exists throughout the program. The main method further demonstrates the functionality of the GameService class by adding and displaying two game instances, then employing a SingletonTester class to affirm the single-instance nature of the GameService class.
7. SingletonTester: The SingletonTester class contains a method, testSingleton(), designed to assess the singleton behavior of the GameService class. The method obtains a local reference to the singleton instance of GameService, ensuring the existence of only one instance.

**"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** | Mac servers are suitable for smaller-scale applications. For "Draw It or Lose It," considering scalability is crucial. While Mac offers a secure environment, it might face challenges in scalability and higher licensing costs for large-scale deployments. Offers server-based deployment with macOS Server, but not typically preferred for scalable web applications due to cost and scalability concerns.  **Costs**: Includes hardware purchases and potential software licensing, not ideal for high-traffic web applications. | Linux is ideal for hosting "Draw It or Lose It" due to its scalability, security, and cost-effectiveness, supporting robust backend solutions like Node.js or Django. These can efficiently manage game logic, including unique name checks and single instance game states. Widely used for its scalability and security, with no licensing fees, making it highly cost-effective.  **Costs**: No licensing fees, ideal for scalable web applications. | Windows Server, with IIS, can support the game efficiently, especially for .NET-based backend solutions, allowing scalable and secure game sessions management. It's well-suited for server-based deployment, especially for .NET applications.  **Costs**: Licensing fees vary by edition and scale, which can be significant for enterprise deployments, necessitating careful budget planning. | Not applicable for server hosting. The focus is on ensuring the game performs well on mobile browsers through responsive design, emphasizing client-side interaction with the server. |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Client Side** | Ensuring the web application is responsive and interactive across Safari and other browsers is crucial, capable of handling real-time updates for team and player management. Responsive design and real-time updates are essential for compatibility, with cross-browser testing tools like BrowserStack recommended to ensure comprehensive compatibility. | Web applications must be responsive and compatible across various Linux browsers, supporting real-time interactions for game management. Emphasis on responsive design for compatibility, with tools like Selenium aiding in ensuring cross-platform functionality. | Development should focus on creating a responsive interface that works seamlessly across Edge, Chrome, and Firefox, enabling dynamic game management. Responsive design principles ensure compatibility across browsers, with cross-browser testing tools essential for a uniform experience. | Responsive and adaptive design is critical, along with real-time updates for team management across iOS and Android browsers. Using frameworks like Bootstrap for responsive design, ensuring seamless experience across devices, with tools like Cordova for testing and real-time game management. |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Development Tools** | Xcode for Mac-specific features when necessary, with Visual Studio Code or JetBrains WebStorm for frontend and backend development. Consider technologies like Node.js for backend logic.  **Costs**: Free for basic use, but Apple Developer Program membership required for distribution (annual fee). Use of cross-platform IDEs is recommended for broader compatibility. | Leverage open-source tools like Visual Studio Code or Eclipse for development, with Node.js ideal for backend game logic.  **Costs**: Mostly free, minimal licensing costs, making it a cost-effective option for game development. Open-source tools minimize licensing costs, with a focus on backend frameworks supporting real-time updates. | Visual Studio provides a robust environment for both frontend and backend development, with .NET Core suitable for backend game logic.  **Costs**: Licensing fees for Professional or Enterprise editions could impact budgeting, with the Community Edition helping mitigate costs for small teams. Visual Studio's Community Edition can help reduce costs, but larger teams may face significant licensing fees. | Cross-platform development frameworks like React Native or Flutter reduce the need for multiple development teams by enabling single-codebase deployment across iOS and Android.  **Costs**: Mostly free, except for specific distribution fees like the Apple Developer Program for iOS app distribution. Use of responsive web design principles ensures seamless access on mobile browsers, with development tools supporting real-time interaction and updates for a dynamic gaming experience. |

## 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**: To facilitate Draw It or Lose It's expansion to other computing environments, we recommend adopting a Linux-based approach. Utilizing Linux allows for leveraging web technologies like HTML, CSS, and JavaScript, which are inherently cross-platform, ensuring seamless deployment across desktop, mobile, and web platforms. This strategy aligns with Linux's open-source nature, promoting accessibility and flexibility across diverse computing environments, crucial for The Gaming Room's goal of multi-platform expansion.
2. **Operating Systems Architectures**: Linux's support for a broad range of architectures, including x86 and ARM, showcases its versatility. This adaptability is essential for accommodating the diverse hardware used in various computing environments, from desktops and servers to mobile devices and embedded systems. Such scalability and compatibility make Linux an ideal choice for The Gaming Room's varied deployment needs.
3. **Storage Management**: For the Linux operating platform, we propose using the ext4 file system. Ext4's capabilities in performance, scalability, and support for large files are well-suited to meet the diverse storage requirements of Draw It or Lose It. Its journaling feature further ensures data integrity and facilitates recovery after system failures, enhancing reliability across computing environments.
4. **Memory Management**: Linux utilizes a virtual memory system that involves dividing physical RAM into smaller portions called pages. Through techniques like demand paging, Linux loads only the necessary portions of the Draw It or Lose It software into memory, minimizing resource usage. Linux also employs mechanisms such as swap space, allowing inactive parts of the software to be temporarily moved to disk when the physical RAM is under pressure. This ensures efficient utilization of available memory resources, which is vital for the performance of Draw It or Lose It across different devices.
5. **Distributed Systems and Networks**: Implementing a distributed software architecture with a central server coordinating game state and facilitating communication between devices addresses the challenge of multi-platform connectivity. Enhancing this architecture with detailed error handling, data synchronization, and redundancy strategies ensures robustness against connectivity issues. Compatibility layers and comprehensive security measures are crucial for secure and seamless data exchange across diverse platforms.
6. **Security**: To safeguard user information across platforms, we leverage Linux's robust security features, including user permissions and access controls, and enforce end-to-end encryption using HTTPS. Strong user authentication mechanisms, including multi-factor authentication, are essential, alongside secure API practices. Regular security audits and updates are planned to maintain high security standards, ensuring the protection of user data against emerging threats and compliance with data protection regulations.