

# 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 |
| --- | --- | --- | --- |
| 1.0 | 07/16/2023 | Tiffany Thai | First iteration |
| 2.0 | 08/06/2023 | Tiffany Thai | Update Evaluation and Recommendations sections |

## [Executive Summary](#_sbfa50wo7nsh)

The purpose of this software design document is to outline the proposed solution for developing the web-based game application, "Draw It or Lose It," for our client, The Gaming Room. The game is inspired by the classic TV game show Win, Lose, or Draw, where teams compete to guess what is being drawn. The solution involves rendering images from a library of stock drawings as clues, allowing multiple teams to participate in four rounds of gameplay. By expanding the game to a web-based platform, it can be accessible across various devices and provide an enhanced gaming experience. The proposed design takes into consideration the client's requirements and aims to deliver a user-friendly and engaging gaming application.

## Requirements

1. The game must be accessible through a web-based platform, ensuring compatibility with different devices and operating systems.
2. Each game should support the participation of one or more teams, with the ability to assign multiple players to each team.
3. Game and team names should be unique to prevent conflicts and allow users to check name availability during team creation.
4. The application should enforce the limitation of a single instance of the game in memory at any given time. This can be achieved by implementing unique identifiers for games, teams, and players.
5. The game rounds should have specific time limits, such as one minute per round, with drawings progressively revealed until fully complete at the 30-second mark.
6. If a team fails to guess the puzzle within the given time, the remaining teams should have a chance to offer one guess each within a 15-second time limit to solve the puzzle.

## [Design Constraints](#_2et92p0)

* Web-based Distributed Environment: The game application must be developed for a web-based platform, which introduces constraints related to network communication, security, and compatibility with various web browsers and devices.
* Unique Names: The system should enforce the uniqueness of game, team, and player names to prevent naming conflicts and provide an optimal user experience when creating and joining games.
* Single Instance Limitation: To ensure the game operates as intended, the design must account for the limitation of only one instance of the game service being active in memory at any given time.

## [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 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.**

The diagram above depicts the classes that exist in the Draw It or Lose It game application system. The Entity class serves as the base superclass for all the entities in the application. It contains common attributes such as 'id' and 'name' that are shared by all subclasses. This implies that every entity in the system will have an identifier and a name. The Game, Team and Player classes are subclasses the extends the Entity superclass. These classes represent the main entities within the game application: a Game consists of multiple Teams, and each Team consists of multiple Players.

The GameService class has a composition relationship with the Game class, indicating that it manages the lifecycle of Game instances and holds references to multiple Game objects. Similarly, the Game class has a composition relationship with the Team class, and the Team class has a composition relationship with the Player class.

The ProgramDriver class is where the main function resides. It serves as the entry point of the application. Within the ProgramDriver class, the GameService singleton instance is created, indicating that there will be only one instance of the GameService class throughout the application. The ProgramDriver class is responsible for adding games, teams, and players using GameService instance. Additionally, the ProgramDriver class has a dependency on the SingletonTester class, as denoted by the <<uses>> arrow.

The UML class diagram demonstrates several object-oriented programming principles, including inheritance, encapsulation, and abstraction. Firstly, the diagram showcases the inheritance relationship between the Entity superclass and its subclasses (Game, Team, and Player). Inheritance enables subclasses to inherit attributes and behaviors from the superclass. For instance, the subclasses utilize the super keyword to access the Entity's constructor function, reducing code duplication and ensuring consistency. The GameService class encapsulates its attributes, such as the constructor and the list of existing games, only providing methods to interact with the data. This approach maintains data privacy, ensuring only one instance gets created, as well as promotes abstraction. Additionally, abstraction is showcased by the way the classes manage games, teams, and players, concealing unnecessary implementation details and focusing on essential interactions.

[**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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Unix-based. Stable and secure foundation for hosting web applications. Robust ecosystem and developer friendly. Relatively higher cost for hardware. Limited scalability compared to Linux and Windows. | Open-source operating system allows for customization, flexibility and wide range of software and tools. Highly scalable, known for its stability and security. Some GUI limitations and hardware compatibility. | Wide range of software compatibility and strong developer ecosystem. Broad hardware support and robust support and documentation. More known security vulnerabilities. | Portability. Touchscreen and gesture-based interaction with limited screen size. Varied hardware capabilities. |
| **Client Side** | Intuitive and user-friendly interface reduces learning curve. Developing and maintaining multiple clients increase development costs, time, and may require diverse expertise. | Free to use and distribute, important to consider other costs (hardware, tools). Has a steeper learning curve. Diverse expertise needed for different clients. | Licensing costs may be higher compared to open-source alternatives. | Important considerations include responsive design and connectivity limitations. Native features such camera, GPS, and push notifications. |
| **Development Tools** | Node.js and JavaScript are commonly used. IDEs such VSCode and XCode. | Rich ecosystem includes IDES such as VSCode, Atom and Sublime Text. Robust command-line interface and package management systems like apt or yum. | C# and .NET framework are popular languages for building Window-based web applications. Visual Studio and JetBrains IDEs. | Kotlin, Swift, and Objective-C besides Java and JavaScript. Android Studio and XCode as well as device emulators and simulators. |

## Recommendations

1. **Operating Platform**: To allow the Gaming Room to expand Draw It or Lose It to other computing environments, adopting a web-based operating platform is recommended. This would enable the game to be accessed and played on various devices with web browsers, such as desktops, laptops, tablets, and smartphones. By leveraging web technologies, the game can reach a broader audience and provide consistence experience across different platforms.
2. **Operating Systems Architectures**: For the chosen web-based operating platform, the architecture will primarily revolve around client-server and web-based technologies. The client-side architecture will involve the use of HTML, CSS, and JavaScript to render the game interface and handle user interactions. On the server-side, a multi-tier architecture can be employed, consisting of presentation, application, and data tiers.
3. **Storage Management**: An appropriate storage management system would a be combination of a relational database management system (RDBMS) and cloud storage services. The RDBMS can handle structured data related to game progress, user profiles, and game statistics. The cloud storage services can be used for storing media assets, such as the stock images, ensuring scalability and accessibility across platforms.
4. **Memory Management**: The recommended web-based operating platform utilizes automatic memory management techniques provided by modern web browsers. Browsers employ garbage collection mechanism to handle memory allocation and deallocation, freeing developers from manual memory management. This helps ensure efficient memory utilization and prevents memory leaks.
5. **Distributed Systems and Networks**: To enable communication between various platforms, the game can utilize distributed software architecture and leverage network connectivity. This can be achieved by implementing a centralized server or cloud infrastructure that serves as a communication hub between game clients. The server can handle game synchronization, real-time updates, and message exchange between players across different devices. The system should account for network connectivity issues, such as intermittent outages or low bandwidth, by having appropriate error handling and synchronization mechanism.
6. **Security**: To protect user information on and between various platforms, several security measures can be implemented. The recommended web-based operating platforms should support secure communication protocols. User authentication and authorization mechanisms, such as username and password, can be implemented to control access to game features and user profiles. Additionally, data encryption techniques can be used to safeguard sensitive user information stored in databases or transmitted across the network.