

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 | 03/23/25 | Rukhsar Amin | The executive summary, design constraints, system architecture view, domain model, cover page, document revision history, and suggestion were all altered. |

**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)

The Gaming Room, our newest customer, is looking to create a web-based game that can run on several different platforms. Currently, Draw It or Lose It is the game they wish to use as a basis. Currently, this game is only compatible with Android; the developer hopes to make it compatible with other operating systems like Windows, Linux, MacOS, and others. In the game, teams compete to guess the images that are being drawn. Both the guessing and the drawing mark time have time constraints.

## Requirements

1: The game needs to be playable on a web-based platform that works with a variety of hardware and operating systems.

2: It should be possible for one or more teams to participate in each game, and each team should be able to allocate several participants.

3: It is recommended that game and team names be distinct in order to avoid disputes and enable users to verify the availability of names when creating a team.

4: The program should enforce that there can only be one instance of the game running in memory at once. Using distinct IDs for players, teams, and games will help achieve this.

5: The game rounds should have time constraints, like one minute each, and the drawings should be presented one at a time until the 30-second mark, when they should be finished.

6: The remaining teams should be given a chance to make one guess each within a 15-second time limit to answer the puzzle if one team is unable to do so in the allotted time.

## [Design Constraints](#_2et92p0)

* The game application needs to be made for a web-based platform, which presents limitations with regard to network connectivity, security, and compatibility across different web browsers and devices.
* Unique Names: To avoid naming disputes and offer the best possible user experience when establishing and joining games, the system should dictate that game, team, and player names must be unique.
* Limitation of Single Instance: The design must factor in the restriction that only one instance of the game service may be running in memory at once in order to guarantee that the game functions as intended.

## [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 classes that make up the Draw It or Lose It game application system are shown in the figure above. Every entity in the program is a base superclass of the Entity class. 'Name' and 'id' are among the common properties that all subclasses share. This suggests that each entity in the system will have a name and an identification. Subclasses that extend the Entity superclass are the Game, Team, and Player classes. Within the game application, these classes stand in for the primary entities: a game is made up of several Teams, each of which is made up of several Players.

In order to manage the lifecycle of Game instances and save references to numerous Game objects, the GameService class has a composition relationship with the Game class. In the same way, there is a composition relationship between the Game class and the Team class, as well as between the Player class and the Team class. The main function is located in the ProgramDriver class. It functions as the application's entrance. The creation of the GameService singleton instance within the ProgramDriver class signifies that the GameService class will only exist in a single instance across the program. Using GameService instances, the ProgramDriver class is in charge of adding games, teams, and players.

The UML class diagram illustrates a number of object-oriented programming concepts, such as abstraction, encapsulation, and inheritance. First, the diagram illustrates how the Entity superclass and its subclasses (Game, Team, and Player) are related through inheritance. Subclasses can acquire traits and behaviors from the superclass through inheritance. For example, the subclasses reduce code duplication and ensure consistency by using the super keyword to access the Entity's constructor function. Its properties, including the constructor and the list of games that are currently available, are contained in the GameService class, which solely offers methods to work with the data. This method ensures that just one instance is produced, protecting data privacy while encouraging abstraction.

**"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** | operating on Unix. secure and reliable platform for web application hosting. stable and developer-friendly ecosystem. Hardware costs are comparatively greater. Scalability is restricted in contrast to Windows and Linux. | In addition to being extremely scalable, robust, and secure, open-source operating systems provide customization, flexibility, and a vast array of software and tools; yet, they also have hardware compatibility problems and GUI constraints. | Although the software has known security flaws, it also has a strong developer community, a wide range of compatibility, and comprehensive device support and documentation. | The gadget has multiple hardware capabilities and a small screen size, but it also offers mobility, touchscreen, and gesture-based interaction. |
| **Client Side** | The learning curve is lowered with an intuitive and user-friendly interface. Creating and sustaining a large clientele adds time and expense to development and may call for a variety of skills. | Although the service is free to use and distribute, it necessitates varying levels of competence for various clients and takes into account other expenses like hardware and tools. | Compared to open-source alternatives, licensing fees could be more expensive. | Limitations in connectivity and responsive design are important factors. Push alerts, GPS, and a camera are examples of native functionality. |
| **Development Tools** | JavaScript and Node.js are frequently utilized.  IDEs like XCode and Visual Studio Code. | The ecosystem includes package management systems like apt or yum, a powerful command-line interface, and IDES like VSCode, Atom, and Sublime Text. | Windows-based web applications are frequently created using the C# and.NET framework, and the Visual Studio and JetBrains IDEs are common options. | In addition to Java and JavaScript, there is Kotlin, Swift, and Objective-C.  Device emulators and simulators, along with Android Studio and XCode. |

## 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 expand Draw It or Lose It to different computer environments, The Gaming Room recommends building a web-based operating platform. This would enable the game to be played on a variety of devices with web browsers, increasing its reach and providing a consistent experience.
2. **Operating Systems Architectures**: The selected web-based platform will make use of web-based and client-server technologies, with a multi-tier server architecture for display, application, and data tiers and HTML, CSS, and JavaScript for client-side interfaces and user interactions.
3. **Storage Management**: In order to handle structured data, such as user profiles and game progress, and to guarantee the scalability and accessibility of media assets, such as stock photos, across platforms, an appropriate storage management system integrates a relational database management system (RDBMS) with cloud storage services.
4. **Memory Management**: The suggested operating system is web-based and makes use of automatic memory management features offered by contemporary web browsers. By using garbage collection mechanisms to control memory allocation and deallocation, browsers relieve developers of the need to manually manage memory.
5. **Distributed Systems and Networks**: By using a centralized server or cloud infrastructure as a communication hub, the game can make use of distributed software design and network connectivity. In addition to managing real-time updates, message exchange, and game synchronization, this server also takes care of network connectivity problems like sporadic outages or limited bandwidth.
6. **Security**: A number of security measures can be put in place to safeguard user data both within and between platforms. There should be secure communication protocols supported by the suggested web-based operating systems. It is possible to restrict access to game features and user profiles by implementing user authentication and authorization techniques, such as login and password. Additionally, private user data that is sent across a network or stored in databases can be protected by using data encryption techniques.