

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 | 09/20/2024 | Kingsley Oti Amponsah | Updating the game with new features such as multiple teams, multiple players, naming of teams and the ability for the game to exist on memory at any given time. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It, which is currently available in an Android app only. This game is played by various teams of many players who compete in four rounds at a time. One team guesses till the allotted time runs out when a photo is selected from a collection of pictures. If no one responds, every member of the other team has 15 seconds to respond.

## Requirements

## [Design Constraints](#_2et92p0)

1. Operating System Environments: The key barrier in constructing a web-based version of this game comes in assuring real-time interaction among several participants across different operating platforms. This calls for:

* Numerous gaming sessions running simultaneously while making effective use of server resources.
* Real-time gaming experience with low latency data transmission.
* interoperability across platforms to reach more people.

1. Resource Allocation: since the existing code for the original “Daw It or Lose It” app is in Java, we may need to acquire tools and resources to be able to help us get the work done in real 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)

For the Game, Team, and Player classes, the Entity class effectively serves as our handler. They get the Entity class's traits from the three. The four classes that were developed—Game, Team, Player, and GameService—all make references to one another. We design or construct the project using the ProgramDriver class to satisfy our clients' needs. We can access and run every class we made from the ProgramDriver class. Additionally, there is the SingletonTester class to consider. Every class that is structured enables the project to proceed in accordance with the design specifications, allowing for the simultaneous use of several teams and players during a single game session.

**"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** | Although license costs are high and Mac books are required for development, Macs may be utilized as servers. Its GUI (Graphic User Interface) is also easy to use | Linux is suitable for use in web hosting scenarios.  The most widely used of all, it requires no authorization at all. It is also cost friendly and difficult to navigate | Servers running on Windows are incredibly easy to set up and operate, and they are also highly secure. But the cost of licensing is high. | While mobile devices are capable of serving as servers, they are not fully optimized for this role. Their power is insufficient for high end. However, it might be used to development. |
| **Client Side** | Very costly for end consumers.  Also requires some experience and moderate time.  Accurate abilities and OS navigation are required. | It takes a great deal of experience and patience.  To utilize the operating system, you must have Linux data. Cost is expensive and you will need an expert in python to manage it. | More costly compared to Linux systems.  Learning how to support a Windows setup is simple.  Minimal level of experience required | You should look for developers with prior expertise creating apps for mobile devices.  Distinctive measures must be made to manage user interaction and display, in contrast to online platforms. |
| **Development Tools** | languages made up of JavaScript, CSS, and HTML.  libraries to help with front-end programming and  additional development tools comprise of PyCharm, Visual Studio, GitHub, and so on. | languages made up of JavaScript, HTML, and CSS.  Libraries for languages and front ends. Python, PHP, Ruby, and JavaScript are all used in Linux systems. | languages made up of JavaScript, CSS, and HTML.  Has Libraries for languages and front ends.  Among the developer tools are PyCharm, Eclipse, command prompt, and so on. | languages made up of JavaScript, HTML, and CSS.  frontend and language support libraries. Programming languages include HTML, PHP, C++, and Python IDEs. |

## Recommendations

1. **Operating Platform**:

Linux would be my choice for the server's operating system, depending on the client's needs. Linux's speed, reliability, and security make it a popular choice for server applications. Because Linux is very configurable and offers great support for web applications, developers may customize the system to meet their own needs. Furthermore, because Linux is open source, a sizable development community is always enhancing and maintaining it. Since the game Draw It or Lose It is web-based, Linux would be a good platform to host it on.

1. **Operating Systems Architectures**: A multi-tier design would be advised for the suggested distributed system. The program's display, application, and data tiers are divided by a multi-tier design, which facilitates better fault tolerance and more effective scaling. While the application tier manages the business logic, the presentation tier renders the user interface, and the data tier oversees data storage.
2. **Storage Management**: For the game to run well, a high-performance storage solution would be needed because it renders graphics from a sizable library of stock drawings. I would advise storing the server's data on a solid-state drive (SSD). Images would load more rapidly on SSDs since they are quicker than conventional hard drives and offer better read and write rates.
3. **Memory Management**: Excellent memory management support is offered by Linux, the suggested operating system. Linux manages memory with a paging mechanism that swaps unused data to disk, enabling effective use of physical memory. The Draw It or Lose It program would benefit from this as it would guarantee optimal memory consumption, which would enhance performance.
4. **Distributed Systems and Networks**: A RESTful API would be suggested in order to facilitate communication across different systems. Widely utilized for online applications, RESTful APIs allow communication over HTTP across the internet. RESTful APIs have great support and are simple to build on Linux, the suggested operating system.
5. **Security**: As with any web-based applications, security is crucial, and Draw It or Lose It should be no different. Excellent support is offered by Linux, the suggested operating system, for security features including access control and firewalls. Furthermore, Linux has a reputation for being less susceptible to security breaches than other operating systems and is well-known for its security. I would advise implementing SSL/TLS encryption for all communications in order to safeguard user data both inside and between platforms. This would guarantee that any data sent back and forth between the client and the server is safe and unreadable by others.