

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 1.0

Dhiraj Gurung

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.2s8eyo1)

[**Evaluation 4**](#_heading=h.17dp8vu)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/26/2025 | Dhiraj Gurung | Project One: Iteration 1 |
| 1.1 | 02/04/2025 | Dhiraj Gurung | Project Two |

**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](#_heading=h.35nkun2)

The client The Gaming Room is looking to develop a web-based application for game Draw It or Lose It. It is currently only available as an Android app but they are looking to make it available for other platforms like Mac, Linux, and Windows. The client is also requesting some key requirements that must be added to the game: the ability to have one or more teams, each team to have multiple players, unique game and team names, and to have only one instance of a game in memory at any given time.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_heading=h.1ksv4uv)

1. **Creating a Cross-Compatible Application:** Since the app will be a web-based application there are some features that may be easier to implement in one browser over the other. For example, some requirements may be easier to implement for Chrome users than it is for Safari users. The implications of creating a cross-compatible application mean that more time and effort will be needed to ensure that a feature works for all of the platforms. Additionally, this also means that there will be an added financial cost as well because more time spent means more money spent paying for the development costs.
2. **Time and Budget:** Although not explicitly stated time and budget are always a constraint for development. Resources are finite and therefore the app development should be done in an efficient manner so as to not waste both time and money with redundant coding and testing. The higher the budget the more time and resources that can be spent trying to refine the app.
3. **Server Size and Long-Term Sustainability:** Since the game will be played by players in real-time across the web there needs to be adequate money spent on servers. This is done not only to host the games but also to handle the number of players playing at any given time without laggy or glitchy gameplay. The active player base is one factor in determining how big the server size is and how much money should be spent to maintain it.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

**"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 UML diagram above shows the classes that exist within the Draw It or Lose It app and how they interact with each other. One object-oriented principle that can be found within the diagram is inheritance because the Entity class serves as a base class for the Game, Team, and Player classes. This means that the Game, Team, and Player classes are derived classes and can access all the protected and public members of the Entity class. An example of run-time polymorphism within the derived classes is the toString function because each of the derived classes has its own version of the toString function which they are overriding from the Entity class.

The GameService class has an association relationship with the Game class and a zero-to-many multiplicity. The same relationship and multiplicity applies to the Game and Team Class as well as the Team and Player class.

The ProgramDriver class is the driver that starts off the program. It uses a singleton class design pattern ensuring that only one instance of the GameService class will be used throughout the whole application. The <<uses>> also indicates a dependency relationship with the SingletonTester class.

## [Evaluation](#_heading=h.z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are also expensive to run and maintain but are easier to configure due to the Mac OS. Will have more apps that are compatible and optimized for Mac than Linux but not as many as Windows. | One of the pros of using a Linux-based server is that it is open-source and free to use. It is also the best option for hosting servers as it is quite reliable and due to its open-source nature, you have more control over how to configure it to best suit your needs. One weakness of the Linux Os is that most apps aren’t optimized for Linux. | The servers are expensive to host but it has a user-friendly operating system and is compatible with most development apps and tools. You will also be faced with licensing fees for using the software but you will get regular security updates automatically. | Mobile devices are smaller by default and thus generally not as powerful as the devices running Mac, Linux, or Windows. Due to the large variance in devices, it’s more difficult to implement features that will be compatible with all mobile devices. You may need to implement features differently to be optimized for the given hardware limitations. |
| **Client Side** | The Mac UI is clean and minimal but you will need to have some expertise to navigate through the operating system. Additionally, it will be expensive for users to use and will require some time to get used to. | While you don’t have to pay for licensing fees with Linux you do need more expertise to use Linux as it is the least user-friendly OS out of the 3 non-mobile-based operating systems. There are a lot of tools available on Linux but you also need a high learning curve to use it effectively | One consideration of using Windows is that you need to pay for the licensing fees, however, it has a smaller learning curve as it is a more user-friendly OS than both Mac and Linux. | There is more variance in hardware and thus more expertise may be needed to implement the features. More time and cost may be required to ensure compatibility with different devices. |
| **Development Tools** | **Languages:**  JavaScript, HTML, CSS, Python, Ruby, Java  **Tools:** VScode, Pycharm, Git Hub, Eclipse | **Languages:**  JavaScript, HTML, CSS, Python, Java  **Tools:** VScode, Pycharm, Git Hub, Eclipse | **Languages:**  JavaScript, HTML, CSS, Python, Java  **Tools:** Tinker, VScode, Pycharm, Git Hub, Jupiter, Eclipse | **Languages:**  JavaScript, HTML, CSS, Python, C++  **Tools:** React Native, Flutter |

## 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**: Windows is the recommended operating platform because of its user-friendly interface and its compatibility with a wide range of development apps. Choosing Windows will allow The Gaming Room access to the platform with the largest user base and the largest amount of developer tools that are easily accessible.
2. **Operating Systems Architectures**: The Windows platform is planned with a general audience in mind and as such you can access the web and start downloading files as soon as you have set up a user profile. In order to accomplish this Windows uses an architecture comprising of the operating system (OS) kernel, system services, and applications. The OS kernel is responsible for handling OS requests, the system services are built into the OS and work automatically without user input, and finally, the applications are installed with user input and must be opened manually.
3. **Storage Management**: I recommend using a cloud service to manage storage as it is the easiest way to set up, especially for a global audience. However, if The Gaming Room does have the resources to both build and maintain their own servers that might be a cheaper option in the long-term.
4. **Memory Management**: Windows uses a paged memory management technique this allows for a more efficient of free space and avoids errors with segmentation that might occur from using a segmented memory management technique.
5. **Distributed Systems and Networks**: Multiple servers will be necessary to balance the loads of a large client base. Using the distributed systems alongside a load-balancing strategy can help ensure better connectivity and having multiple servers to shift the load around can prevent any one server from being overloaded. A cloud-based server provider is one option for starting quickly however you may have to pay more for the convenience of using someone else’s server architecture.
6. **Security**: Windows does come with its own built-in security features that warn the user from installing malware or third-party apps. However, I also recommend using encryption to secure any data sent to and from the user. Also, implementing strict and industry-standard password security protocols like 2-factor authentication can also help reduce data breaches. Implementing multiple security measures is the best way to keep the application safe and secure.