

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/23/22> | Lance Cain | Adding Executive summary, design constraints, and updating UML diagram |

**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)

Draw It or Lose It is a game of guessing that has multiple rounds with a timer and the players have to guess what an image slowly rendering may be. If it is not guessed by the current team then other teams will get a chance to guess with a limited time. This software design template is intended to make the currently existing game cross-platform.

## [Design Constraints](#_2et92p0)

Any quantity of teams involved

A team can have any number of players

Unique game and team names to prevent duplicates

## [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)

Demonstrated in this UML Diagram are a few objected oriented principles such as inheritance between the entity superclass and the game, team, and player subclasses which extend the entity class. We then have lines demonstrating association between the gameservice, game, team, and player classes as well. The ProgramDriver is showing dependency on the SingletonTester as well.

**"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** | As of April 21, 2022 macOs Server has been discontinued, likely due to apple products being user friendly which is contradictory to security and functionality of a server | Linux provides a completely configurable and functional system that once properly setup is secure and stable. | Windows platform provides a reasonably stable and up to date platform for hosting that can host most any language and functionality of a web based application. | Mobile devices are poor for hosting applications due to their specialized use case design of being an interface with a large screen and an unstable wireless connection for data. |
| **Client Side** | Down sides being extra licensing fees and code reviews, limited library availability and proprietary systems. Pros being very user friendly and elegant design. | Linux has the worst client experience that can be difficult to make consistent and smooth between clients, however systems in place tend to maintain stability for a longer operational life. | The client side windows experience was the most common interface until recent years and provides a functional and capable platform with decent security with minimal effort from the client. | Client side applications for phones can be trickier to achieve good results however when possible will reach the largest audience and provide the most available experience to clients |
| **Development Tools** | Apple has comprehensive tools that are provided for development on their hardware and the preferred development languages. However the variety of tools is smaller due to the licensing required by the platform. | There is no shortage of every imagine combination of development tools available for linux platforms as well as tools to develop for other platforms being readily available with communities and guides galore. | Development tools for windows platforms are available for most any platform and with even some languages allowing for applications to be written in any text editor and compiled, has virtually no limits. | While there are some development tools available for mobile platforms, most development is done on windows or linux computers for mobile devices which does have a broad range of development kits and languages available. |

## 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**: Using a Client-Server computing platform allows for client devices of a larger variety to connect to more specialized servers that are more efficient and capable of handling the necessary demand as well as interconnectivity and scalability for multiplayer capability. While peer to peer can work for game applications, it is often less reliable and more difficult to implement for no main stream benefit. A client-server platform can be more easily scaled into a web based environment such as different cloud services are frequently done today.
2. **Operating Systems Architectures**: Linux would be the preferred server operating system due to it’s adequate security, stability, and ability to server on virtually any device in existence that is meant to have an operating system. On the client side we would create a web client that was accessed through a web browser which would allow clients of any fully web capable operating system to access the game.
3. **Storage Management**: Persistent storage on the server side would be done with MySQL since it is the most widely used and is open source as well as royalty free. This will give persistent storage of client data that is accessible for any number of servers and can also be scaled as well. The client side will store any necessary data for login credentials in a cookie to be used in authentication upon return to the website or app allowing us to download any needed information to maintain the illusion of data stored locally on the device.
4. **Memory Management**: For both the server and client side we will maintain the focus of buffers and not storing unnecessary data outside of the needed scope. Reserving the anticipated memory required for storing the rendering images as well as future rendering images within a small time window will allow us to keep memory consumption to a minimum allowing most any modern day device the ability to play our game.
5. **Distributed Systems and Networks**: Using a client-server operating platform we can achieve some redundancy as well as allow scalability to combat sudden or rapid changes in demand. Having servers that can spin up and fill a new demand because we released a new image pack for the game bringing in a new flock of bored lunch staffers looking to escape their colleagues rants about the latest headline news, is crucial to our games success. Having servers in different locations allows for internet backbones to route our traffic to the available server if one of them were to go offline for some reason such as weather or nation state threat actors.
6. **Security**: Security is crucial to any software solution’s success. With Linux as our server operating system in a client-server environment, and strict information access and control, we can maintain a stable environment. With redundancy and regular monitoring of servers we can minimize data breaches and inconsistencies in client experiences. Storing user data only on the server allows us to avoid many of the security flaws of devices such as iphones or android devices which are often overlooked and can give a large hole in security barriers put in place.