

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

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

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/17/24 | Zach Shires | I updated the project’s summary, requirements, and constraints to match the new goals. |
| 1.1 | 12/1/24 | Zach Shires | I Identified strong/weak points of potential hosting platforms. |
| 1.2 | 12/15/24 | Zach Shires | I added recommendations on how to proceed with the project development. |

**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](#_35nkun2)

The Gaming Room is currently making a new game called Lose It or Draw It. In this game, teams will compete to guess what is drawn from a large library of stock images. Each game will consist of four one-minute rounds with the drawings being rendered for 30 seconds until completion. The client wishes for the development to be in a web-based environment and for the capability of multiple unique teams and games. We will also need to create it so that only one instance of the game can exist in memory at a time.

## Requirements

The Gaming Room requires the game to have the ability to have one or more teams in the game, and multiple players assigned to the team. The game IDs and team names must also be unique so we will have to check for that before allowing any of them to be created. There also can only be one instance of the game in memory at any time.

## [Design Constraints](#_1ksv4uv)

As a web-based application, the game will need to be compatible with multiple platforms. The implications of this are that we will either need a cross-platform development framework or separate codebases to ensure compatibility. The second option will also increase the overall scope of the project and could strain resources.

## [System Architecture View](#_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](#_2jxsxqh)

The Entity class is a parent to the Game, Team, and Player class. This means those three classes inherit any attribute it contains. The Game class is associated with the Team class in a zero-to-many relationship. The same is true for the GameService class and also how a Team object is associated with a Player object. The GameService class is designed as a singleton so only one instance of it can exist. The id and name attributes showcase encapsulation as they can only be accessed through getter methods. The toString() method also showcases polymorphism as it is overwritten in Game, Team, Player, and Entity. Last, the Program Driver calls the main method and uses the SignletonTester.

**"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](#_z337ya)

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** | Utilizing a Mac to host your software application has the benefit of strong integration between Apple devices as the ecosystem is designed with that in mind. That also comes with the increased cost of purchasing Mac hardware to run the server. The specific server OS has also now been discontinued and integrated into the regular OS, leaving some of the advanced features behind. | Linux is an open-source OS that has a variety of distributions to choose from. It tends to be more cost-effective, and since it is so widely used, there is a lot of community support. There can also be a steeper learning curve as it relies heavily on the CLI and some software doesn’t have native support. | Windows Server has a very user-friendly GUI as well as integration with other Microsoft products such as Azure. There tends to be higher licensing costs with this solution, as well as slightly worse resource utilization. | Mobile devices have less powerful hardware than the other options, making them only useful for small-scale web apps. They also lack the enterprise security features of their counterparts. They are readily accessible, however, and can make for good options for testing and learning. |
| **Client Side** | For developing on a Mac, extra time will need to be allocated to accommodate approval from Apple’s app store as well as to ensure the program adheres to their HIG guidelines. Hardware costs will be higher as the price-to-performance ratio tends to be higher. Also, there are specific Apple development skills you will need such as understanding native macOS APIs. | Linux can be an affordable option as it is compatible with a larger array of hardware, allowing you to scale the hardware as needed. However, you will need more specialized expertise in things like the CLI and shell scripting. Also, taking into account the different package managers across the distributions can increase the development time. | Windows Development tools such as Visual Studio might require a licensing fee, increasing the cost. You also have to test and debug for multiple versions of Windows as older builds are still popular. There is also a whole suite of Windows software tools where if you choose to use them, you will need that specialized knowledge. | For mobile development, two different OSs will need to be accounted for. You will need someone with expertise in Android development as well as iOS. This will either increase cost as you hire more developers or increase time as you work on them sequentially. |
| **Development Tools** | Xcode is the official IDE for Mac development, but most of the popular third-party programs exist as well. You will also find a lot of programs written in Swift, as well as some legacy programs written in Objective-C. | Visual Studio Code is a popular lightweight option for Linux. This is a free program and also contributes to the low cost of running your program on Linux. Java, Javascript, and Python are three popular languages for server development as well. | Visual Studio is a powerful IDE created by Microsoft, but you can use third-party options such as Eclipse or Visual Studio Code. Java and Python are popular options for languages as well, but Windows also has the .NET framework where C# is widely used. | For mobile app development, you will probably be using languages like Swift for Xcode and Kotlin or Java for Android Studio. For backend development Python, Java, and Node.JS are popular options as well. There is also a small licensing fee to get into the developer programs for each OS. |

## 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**:

I would recommend an operating platform that is based on Linux, so perhaps Ubuntu Server. It is open-source which will reduce the licensing costs and also has a large library of free software. Ubuntu is also known for being very scalable and can support a wide range of runtime environments, helping with cross-platform compatibility.

1. **Operating Systems Architectures**:

Linux has a monolithic kernel, so all of the core functions are integrated into one large program. The kernel controls things like the device drivers, as well as memory and process management. There is also the user space where you have programs to interact with the kernel but in a much more secure way.

1. **Storage Management**:

I would recommend a cloud storage management system, so something like a storage system from AWS. It can be scaled alongside the user base of the game and can reduce the initial start-up cost of getting the game running.

1. **Memory Management**:

Caching is one technique that will be useful here to reduce image retrieval times. Rather than loading images from the hard drive, we can cache the upcoming images in memory. Linux also has the capability to use virtual memory, so if there are any large workloads where we need to swap inactive data to the hard drive we can. The game should also be developed with a garbage collection mechanism to keep the memory from leaking as the game progresses.

1. **Distributed Systems and Networks**:

Since the game will be a client-server model with the server hosted on something like AWS, there needs to be a continuous real-time connection. The game will need a reconnection logic to handle any short disconnections and get the user back in the game. We should also have redundant servers to handle any outages without interrupting the gameplay. We will also need a form of load balancing to handle the requests for our multiple servers so there aren’t any bottlenecks of specific ones.

1. **Security**:

We should have a role-based access control to separate different types of user’s access to sensitive data. We should also make sure all the user data in the database is encrypted. AWS has tools built in that can do both of these, which is partly why I think it’s a good option. AWS also has protection from things like DDoS attacks and active security services that can detect for malicious behavior.