

# 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 | 09/18/22 | Ronald Sudol III | Completing Executive Summary, Design Constraints, and Domain Model |
| 1.1 | 10/02/22 | Ronald Sudol III | Evaluation Table |
| 1.2 | 10/16/22 | Ronald Sudol III | Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

We have been contacted by The Gaming Room to facilitate the development of the web-based version of their already popular mobile gaming app, Draw it or Lose it. The Gaming Room requires that the web-version be available across all platforms to attract the broadest possible market of users. They also require that the game support multiple teams with multiple players. Further, the game instances, team names, and player names should all be unique with only one instance of each in memory at a time. We will accomplish this by employing a singleton design pattern and iterations patterns to keep track of names and instances already stored in memory.

## [Design Constraints](#_2et92p0)

Cross-platform functionality is important to this project so the program must be deigned in a way that can interact with Windows, Mac, and Linux, as well as it’s already available mobile format.

Storage and memory constraints also need to be kept in mind as the game will be utilizing a large bank of stock images to be used in the game. The game will also be storing player and teams names as well as named game instances and ID numbers. Program should be designed in as compact a way as possible to save room for these data points and images in memory.

Speed of execution will also be a considerable design constraint as players and teams will be playing with each other in real time. The Gaming Room’s server will need to be up to par to handle this. Our program should also be designed in a way that executes the games functions as quickly as possible to reduce the chance of a noticeable time difference between functions on the user’s end.

As always, security is an important design constraint. With a large number of users accessing the game vie the internet, it is imperative that our program design protects individual user data as well as The Gaming Room’s data.

## [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)

Two important classes that operate this program are a driver class that holds the main() function and a singleton tester class that ensures only one instance of a GameService() is created at a time. There is also an entity class that lends its attribute through inheritance to the player and team classes.

Once a single instance of a GameService() is created, the program checks for an instance of a game() by name or ID number and if that name and game number doesn’t exist yet, and instance of game() is created. Next, game checks for an existence of team() and if that team() doesn’t exist, one is created. Finally, the instance of team() checks for the existence of a player() and if that player doesn’t exist yet, one is created. Utilizing iteration to check for already existent instances before creating a new instance ensures that only one unique instance of a given name or ID can be saved in memory at a time.

Polymorphism and Inheritance are present in the way that multiple classes are in fact extensions of the entity class. Further, encapsulation is utilized to not only protect data stored in memory, but to ensure that a player() cannot be created without a team(), and a team() cannot be created without a game() by burying the constructor call that creates on instance in the class that is created before. In this way a GameService() must be created first, then a game(), then a team(), then a player().

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac can be used for a server but it requires licensing and particular hardware. Licensing, hardware, and hosting can all add up making Mac an expensive option for server side of a web game. | Linux is a popular option for web-based server side applications. It is a free open-source option so licensing and fees will be minimal. Linus is also compatible with premium cloud services like Google or Amazon. | Windows is a great option because it has a familiar graphical interface that will be second nature to many users. Windows also requires licensing and hosting, so it may be a desirable option but it can also become an expensive one. | Given their more limited nature in terms of memory and processing, mobile devices are not ideal for hosting server-side web applications. Mobile platform would not be able to handle the amount of traffic generated by a multi-user server. |
| **Client Side** | Mac user’s make up about 16% of the user market. Mac hardware is required for the implementation of client side programs so having a Mac-ready option is advisable but should not necessarily be the focus for client-side development. | Development using Linux is versatile because it can be done in C++, Java, or Python. There are countless free IDEs like Eclipse and PyCharm that but used. While Linux may be a very cost effective option, it’s not the most widely used. | Windows is the largest market percentage, representing about 75% of all users. There may be issues around licensing that can become expensive but Window’s users are the largest group of client-side users that can be reached. | Mobile users typically do not share devices or platforms so development here requires the application be formatted to the user’s device. Android and iOS would each need their own version of the mobile application so development here could be more time consuming but can be addressed by ensuring code is portable. |
| **Development Tools** | XCode is a common Mac IDE which would require licensing fees and supports development in Objective C and SWIFT languages. | PyCharm and Eclipse are free IDE’s that support the use of C++, Java, and PyThon and are compatable with Linux operating systems. | Visual Studio is a popular and useful tool for Windows Development in C# and .NET languages. Visual Studio would require licensing | Android Studio is a free and effective option for developing mobile applications. For people with Apple phones, iOS applications can be developed with XCode for a fee. |

## Recommendations

1. **Operating Platform**: Linux is an efficient and cost effective option for game development. Because it is an open source operating platform there will be now licensing fees. Its familiar graphic interface can be mastered rather quickly by any developer familiar with other platforms. Linux is very customizable and utilizes minimal system resources so it can quickly and effectively be used for a wide variety of applications.
2. **Operating Systems Architectures**: The system architecture should utilize a client and server model. Because the game should be available to the widest variety of devices and browsers, the client side will consist of front-end apps tailored to individual devices. The server side will back end processing and will handle storage of individual game services, large data sets like the images or names of players, and authentication and singleton patterns. It is likely best to two servers on the server side, one as a data server for handling long term storage like authentication information as well as storage of game session and player name instances, and the other server will work as a moderator by receiving and transferring requests between the user’s end and the data serve.
3. **Storage Management**: A separate data server should be utilized for storing user authentication information, game service instances and player and team names. This data server will only be accessible by the web server which helps to protect information contained within. Because the data server is a separate physical entity, upgrading and maintaining is simplified to only working with that individual server.
4. **Memory Management**: The Web server will need considerable RAM capabilities because it will be handling a considerable number of temporary data points and requests being transferred between the data server and user’s device or browser. Linux’s minimal use of system resources should help keep this efficient.
5. **Distributed Systems and Networks**: The web server will moderate and interpret requests from various user devices. While the user’s devices will have their own apps, the communication with webserver should be as uniform as possible. The web server can than communicate directly with the data server in a strictly uniform manner. Having the web server as a moderator allows for the individual updating of user apps as well. For instance, if an update or patch was needed for Android devices, it could be implemented on the client’s side without effecting the web server too much and without interfering with IPhone apps at all. The main concern in terms of outages will be when maintenance or updates must be done on the web server itself. This type of maintenance should be scheduled for times with low web traffic ideally. Updating or perhaps even adding additional storage space to the data server can likely be done executed without effecting the web server too much.
6. **Security**: Encapsulation and authentication are the two key things to focus on here. Making sure that data is only accessible by the right parts of the system should be a focus while coding. Ensuring that user’s permissions follow the principle of least privilege is key. Requiring adequate user authentication, ensuring only a single instance of a session exists at a time, and restricting user access to only the functions necessary to playing the game will ensure that user and system data remains secure.