

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)3-6

[**Recommendations**](#_m8aleynsvzvc)7

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/13/22 | Richard Tetreault | Replaced the bracketed texts in the Executive Summary, the Design Constraints, and the Domain Model |
| 2.0 | 11/27/22 | Richard Tetreault | Updated the table under evaluation; Updated table of contents to better reflect document |
| 3.0 | 12/11/22 | Richard Tetreault | Updated Recommendations |

**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)

To best solve the problem presented, the software needs to have the following functionality. First, it needs to provide a web-based environment where multiple teams, each with multiple players, can be assigned to an instance of the game *Draw it or Lose it*. Second, each team and game name need to be unique for to allow for easy identification. Finally, only one instance of the game can be utilizing system memory at any given time by using unique identifiers for each game.

## [Design Constraints](#_2et92p0)

* The system that the server is running on needs to at least meet minimum requirements of the *Draw it or Lose it* game. As the game is currently only supported on the Android OS, the web-based environment also needs to at least be able to run on the Android OS.
* The server needs to allow for multithreading with a many-to-one model to allow for multiple teams to interact with a single game instance. This means that, while it would be efficient to use, any attempts to have multiple system calls would cause a block until that process is completed. As such, only one game instance should be running per server.
* Only one game can only use system allocated memory. While the game service allows the creation of multiple games, only one would be able to run at any given time. Any new instances of the game being run would stop all other instances until that game’s process is complete.

## [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)

Looking at the gamingroom project UML, each block represents a class the program will use to fulfill its function. Starting with the ProgramDriver, this class contains the main() function to drive the program. The ProgramDriver and SingletonTester classes use each other to function, where the SingletonTester runs code to verify that the GameService is operating using a singleton. The GameService class has a one-to-many relation to the Game class, as there should only be one GameService instantiated while the program is running, while there can be many unique games functioning at the same time. The Game class has this relation with the Team class, and the Team class has this relation to the Player class. Finally, the Game, Team, and player classes all inherit functionality from the Entity class which reduces repetition in code. In other words, we do not need to recreate the code used for the Entity class in each class that inherits from it, which makes each object easier to read as well as more efficient to write.

**"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 OS X is good for hosting a server for Mac devices. It is highly specialized to be able to be efficient with its resources among Mac devices, and it is tough for viruses to be developed for it, making it relatively secure. However, it is pricey, and it being specialized makes it tough if not impossible for clients of other operating systems to run on the server. | Creating a Linux server is free as it is an open-source platform. Linux also is highly secure and has a large community of support. However, Linux requires much more specialized knowledge for effective use including an experienced administrator that can use command line interfaces. | Windows servers allow for easy management for software use. This would include authentication tools, managing permissions, network management, and broad customer support. Windows servers however are expensive to run due to higher infrastructure costs and are vulnerable to attackers due to their relative popularity. Also requires license for continued use. | Since the game is html, it would be best to run it on a pc for use through a mobile device. Using a clous based service such as AWS allows the service to run for a monthly service cost based on usage. |
| **Client Side** | Mac server has a high start up cost and monthly cost. Needs to use specific products for the design, however it is easy to set up and maintain. | As Linux is mainly open source, cost of allowing multiple devices to hook up client side is mainly taken up with system and power costs. Linux requires an experienced administrator to effectively run. | Windows server costs are based on time and use. The more users and data that is recorded, the higher the cost. It is very customizable to the needs of the user. Windows is very widely used and has great product support. | Running on a cloud-based service such as AWS means the only costs would be for the service itself. The cost would change based on the efficiency of the coding as well as the size of the user base. Would require someone to be able to effectively manage memory usage to keep costs to a minimum. |
| **Development Tools** | As the game is HTML, it would be good to use JavaScript to design the environment on Mac. Visual Code Studio would be a good option for an IDE. | Java would be a strong option for developing the app through Linux. Eclipse is a good option for an IDE. | JavaScript would be the go-to for use for developing an HTML game environment through windows due to its versatility. It is widely used and would be easy to find support. | Good to use Java as a language using eclipse as an IDE for development. Java is relatively universal and can be implemented on most OS, including iOS. Python would also be a viable option as the game is HTML. |

## 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 Windows to be the operating platform for the gaming room. This platform allows many supported options to ensure long term operation and the ability to develop the game for cross platform use, such as developing with .NET and the use of Azure servers.
2. **Operating Systems Architectures**: Windows gives the developer a few tools that allows for easy development regardless of platform used. For one, it has access to visual studio and .NET, which allows the developer to code for use among multiple platforms. Windows also allows the use of Azure PlayFab for storage needs of the game service and allows for multiple platforms to be used as clients.
3. **Storage Management**: Using the cloud of Windows Azure PlayFab gives a secure cloud server that can store the information of thousands of players. The cloud is a virtual machine where through an http connection can be accessed through any machine. This would allow players to connect from anywhere and be able to store their player information through the game service.
4. **Memory Management**: Azure PlayFab on Windows allows for scaling of the game service. This means that the number of people connected can be malleable depending on the popularity of the game. As the game grows, this number of people that can connect to the server can be increased as thresholds are met. This scalability can help keep costs to a minimum so space isn’t being wasted on players that aren’t there.
5. **Distributed Systems and Networks**: Azure PlayFab of Windows allows users of the game service to communicate with the server regardless of client platform. It supports everything from android, iOS, Windows, Xbox, Nintendo, and more. All server information is stored in the cloud for use by clients regardless of platform and is free up to ten thousand players on startup, with pricing options if the game generates more users. It also allows for the ability to use data centers in selected regions, so the audience can be targeted based on user localization, minimizing latency.
6. **Security**: As security is an important part of any application regardless of chosen operating platform, there are multiple recommendations that should be followed to ensure safe use. First, ensure that the minimum authorizations are given to each user, such that only the functions that are required for the user to complete their tasks are accessible. This means that game users should only have access to their accounts, teams have access to their teams, administrators access to be able to keep the server operational. Next, authentication should be in use to verify each account. This can be done using passwords in conjunction with a passkey authenticator to minimize risk of stolen accounts. In addition, making sure that network communication is done through a secure transport will also make it difficult for bad actors to infiltrate.