

Draw it or lose it

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 2**](#_Toc115077320)

[**Requirements 2**](#_Toc115077321)

[**Design Constraints 2**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 3**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/18/2025 | Mike Catsoulis | Initial version |
| 2.0 | 08/02/2025 | Mike Catsoulis | Updated evaluation table |
| 3.0 | 8/15/2025 | Mike Catsoulis | Completed recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

Game Room is looking to add to its portfolio a new web-based game across multiple platforms called Draw it or lose it. The application must include multiple users across multiple teams and real time data collection and processing. The solution will follow industry standard programming practices and shall utilize the singleton design patterns.

## Requirements

Multiple platform usability, Multiple user usability during game play, Web based game play and development utilizing Java

## [Design Constraints](#_2et92p0)

1. Scaling – While creating a program that supports only a few teams with a few players can be easier to generate in a short period of time, generating a program that can handle hundreds or even thousands of users and teams requires much more deliberate management of resources and may even require multiple servers.
2. Usability – Creating a program to run across multiple platforms requires much more deliberate development that handles all corner cases that relate to multi-platform use, such as platform specific packages when building the code.
3. Instantiation – The program must only use one instance of game at a time which requires the deliberate programming to keep consistent game state and keeping sync to multiple users on different platforms.

## [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)

The entity is the parent to game, team and player. Contains the common Id and player. Game Service oversees the game and makes sure there is a singular instance. Game Service also tracks Id’s for the game. Games contains a list of teams. Team contains a list of players. Player is the list of the actual users.

Object oriented principles used are inheritance. Entity has shared use with game, team, and player. Additionally the use of private getters and setters are indicative of encapsulation.

**"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** | There is support for server dev on mac with Apache making server deployment possible but not efficient due to high hardware and licensing costs. Mac have been built with the end user in mind mainly focused on user experience. The Mac platform may be used for development of programs but as server due to the points made above such as licensing cost and expensive hardware not really feasible. Additionally open-source support is limited on the mac platform compared to other platforms such as Linux and windows. | Linux is flexible and more utilized due to the support of major web servers. Apache to name one is supported for server development. Linux offers no licensing cost and low hardware cost compared to mac. Linux has a great deal of online support and open-source software packages for any size of server development. Another major win for Linux is security, due to the user model where Linux requires the need for the root password to preform admin style actions as windows if not configured correctly allows any user to run as admin. | Windows supports server applications from .NET applications that are windows based and common server applications such as Apache. Windows provides a GUI as depending on Linux setup it is command line based with no GUI interface, this provides an easier learning experience for newer users. Another consideration is windows has licensing cost which can be expensive depending on the server size and number of servers. Also one other major consideration for windows is the updates, as windows constantly does updates this may pose compatibility issues if a windows update comes and may break the program if there a different drivers for an example | Mobiles not really meant to be a server as they are meant for user interaction they display the front end not process back end. In the Gaming Room application an external server solution must be implemented to server the android clients so a mobile device is not going to work as a server side solution. |
| **Client Side** | The support for all modern browsers, again the limitation will be cost not only on the hardware and software side but not mentioned above due to not being a viable option is the cost of a developer which requires a more specialized skill set of development for mac ios specific. A strong quality check must be completed to provide the all users with the same experience as safari users may have a different experience from other common browser types. | Linus supports modern browser types, but the user interface is much less popular and requires much more extensive knowledge. Still low cost but provides subpar user experience compared to mac and windows. Utilizing Linux requires testing to validate the user experience and depending on the end user device tailoring may be required to get the desired effect across different end user platforms. | Windows supports most modern browsers for ease of accessibility. Due to windows being the go-to for client side there is a large base of developers and a large online presence for open source information and premade software. Due to the accessibility of online information and a large pool of dev’s this makes windows a low-cost solution overall to provide a high quality end user/client side experience. Testing the platform specific use cases provides validation that the client-side specific device works as expected and using windows this is generally expected to work with less client side device specific testing. | To support android and ios developers should use responsive web design to ensure web pages render on various devices. Additionally andoird and ios client side must support the touch feature as this is common on all phones in todays world. Again for ios and android testing is a must, however testing in this scenario can be at a higher cost due to the amount of devices and browsers that are in the current market space. Testing to ensure consistent rendering to varying screen sizes comes at a cost however if there is a business justification it may be worth the cost. |
| **Development Tools** | Eclipse, NetBeans, Java, python. Additionally xcode must be used to build and test apps on mac. Another must have is a developer who specifically deals with mac, as a windows dev who has never developed with mac will have a large learning curve and will slow the dev process down. Another consideration is some platform tool may require paid licenses which may reduce the appetite to use mac systems. | Eclipse, NetBeans python, CLI. Most major tools for Linux are open source and require no purchasing. Linux however requires a more experience developer with prior Linux expertise as Linux is more of a terminal based tool meaning typically does not utilize a GUI interface which has its challenges to a newer dev who has not typically used Linux. | Eclipse, visual studio, python. Windows is maybe the most common platform due to both front end and back-end tools and web based information readily available. Additionally with barrier of entry for learning and becoming proficient the pool of qualified developers is much larger than more Linux and mac specific quality developers. | Android studio, Xcode. If creating an application for both android and ios multiple testing environments are required to test functionality. Developing with these tools is necessary if phone applications are required. Specific developers with specific proficiencies are needed to maintain high quality output and reduce cost by maintaining the deadline without needing to learn new dev tools. |

## 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**: For the development and deployment of the server side of the application Linux is preferred. From a perspective of cost Linux offers low hardware cost and offers a large array of open-source software minimizing operational spin up costs. Linux offers flexibility and reliability for scale and multiple users.
2. **Operating Systems Architectures**: Linux offers a small compact system that operates on a single core. This type of system allows us to install and or remove any necessary or unnecessary files or programs. By using this type of system, the Linux system can be tailored to the “Draw it or Lose it” system architecture allowing for a streamlined, fast and efficient operating system that only contains the necessary programs for full functionality. The primary benefit is smooth operation of the central system giving fast response times and a lightweight solution for easy deployment. Another major benefits is that Linux systems can be imaged easily and transferred to another Linux machine providing easier scalability.
3. **Storage Management**: Linux is compatible with many different file formats making it an easy choice for storage management. Linux has the capability to be hard-wired and connected to a server for storage and or processing online or offline. Linux also has the capability to connect cloud storage areas making Linux for storage management an excellent choice for growing solutions with the need for constant dynamic storage needs.
4. **Memory Management**: Understanding the fact that Linux may be operated without a GUI gives an edge in memory management, and especially in our Draw it or Lose it game. The servers can be operated without the need for a display thus reducing the memory capacity. Additionally in the Linux system the root user can customize how memory is used as windows systems this is done automatically. Lastly Linux has limited if any background processing that takes up CPU utilization wasting the memory and clogging up processing speed. This makes Linux the optimal choice for memory management.
5. **Distributed Systems and Networks**: To enable the requested function of multiple player game in real time Linux is again the best choice for this application. Linux can be programmed to operate in a distributed environment to support the work over multiple servers. Distributing work over multiple servers is essential for scaling when necessary to support the growth of the user base.
6. **Security**: Security is a multi-layer approach with different security features to combat and protect. Some must haves for security is access control, this means who can access what features. For example, controlling the access means game users will not and should not have access to the servers where all the player data is stored. Another must have feature is login in control, this is where each individual user has their own credentials to have only one instance of a certain user. This login in control should have passwords encrypted for even greater security. Another must have for security is a game and security maintenance where there are periodic updates to improve security and as bugs or potential security risks are found regular patching will occur.