

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

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

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

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

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

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

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/19/2023 | Joshua Donnelly | Initial write up |

**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)

The Gaming Room has come to us for a solution to be able to develop their game that serves multiple platforms. Our solution is a game service manager run as a service to manage the server side logistics and to ensure the program can run between multiple systems. Using the singleton pattern we can ensure that there is only one instance of the service running and manage unique identifiers for players teams and games. Using the iterator pattern, we can utilize multiple instances of games teams and players within arrays.

## Requirements

*Hardware capable of running software server side to handle the number of games instanced at any one time.*

*Networking is required to connect players to their teams and to the games.*

*Client-side application will need enough resources to run the game.*

## [Design Constraints](#_2et92p0)

*The more players that are using the game at once the greater the demand on the server. By only allowing one instance of a server at any one time, the number of games will be reliant on the ability for the server to hand out and manage game instances.*

*Operating System constraints will prevent cross over gameplay on different systems without modification, but should not prove difficult to instate.*

*At time of writing there is no effective way to terminate and reuse ID numbers for future games.*

## [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)

Entity is the parent class of Game, Player, and Team. Entity holds the private variables of id and name as well as an overload for toString() and getter methods for those private variables. These methods are used to pass to the children as they each need these methods and variables. Overloading the toString() method by each child class allows the printing to change to the required output for each class individually. Overloading the constructors allows the children classes to use the parent variables. GameService is the server-side requirement to manage instances of games and ensure that there is only one GameService able to run at any one time using the singleton pattern. The iteration pattern was used to manage all of the 0 to many games and ensure the correct instance of the game is managed. Each Game holds an array list of teams for that game there can be 0 or many teams for each game. Each team has an array list of players for that team and there can be 0 or many players for each team.

**"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** | Unix based system that is primarily a client-side platform. Not open-source and requires licensing. Primarily uses a GUI for navigation. | Open-source and based on a Unix like operating system. Tends to support older software longer and is the leading operating system used in cloud platforms. A great server-side platform. | Windows is primarily a client-side platform and is meant for home users. Primarily uses a GUI for navigation. Windows Server is a server based platform and is also not an open-source operating system and requires licensing. | Limited in hardware capabilities primarily used in power saving and portable situations not intended for server side capabilities. |
| **Client Side** | Mac is a very closed eco system and will run applications accordingly. May require extra effort to ensure compatibility with Mac policies. | Many different Linux versions and more being created all the time. Most applications will run fine, but may have to consider differences in OS versions. | Windows applications should be able to run any program developed for Windows even with many different hardware and I/O differences. | Higher hurdle for distribution by having to use application stores, but the fastest way to spread applications especially games. Really only require Android and Apple versions. |
| **Development Tools** | Primarily uses Python for development, but there are many other different IDEs and languages available for use to create applications. | Primarily uses C to code in Linux, but can use many other types of languages as well. | Primarily uses C++ for development, but there are many other different IDEs available for use top create windows applications. Windows encourages application development and even provides some tools for development in their client software. | Java is a great language to use and for iOS and Android applications and can be used for cross compatibility there are tools that can create code for both systems simultaneously making conversion a bit easier. |

## 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**: Linux for server and development and Mobile OS for client side to start with. This will allow for cheaper development based on the open-source operating systems and still provide a wide avenue for people to play and share the game. The client-side application being on Mobile platforms allows for the largest potential for gaining users. Mobile is great for smaller games that are managed more on the server-side platform and have marketplaces for applications allowing users to recommend and share the game with other friends.
2. **Operating Systems Architectures**: Linux is a Unix like environment and can be used even from home computers. The open source and free operating system, makes it easy to get running quickly and efficiently. The lack of licensing costs and the availability for technical support is much higher for server applications. The lack of a GUI also helps with memory and storage management. Mobile platforms are power saving operating systems that tend to run with limited resources. They are very often used for games that are able to be played alone or with friends. Higher demanding games may have issues on Mobile operating systems, but the relatively small demand for this particular application seem to be inline for the resources available for mobile devices.
3. **Storage Management**: This particular game has a large image library that would likely use up too much memory to be used in that manner. Normal storage is much slower than memory, but has much higher capacity. This means that while there may be necessary instances for loading information that the users will notice due to the lower speed of storage, this can still be manageable by using storage as the medium to store and load these larger images. There are still limitations to this type of resource as well and there are ways to compress larger files that may not always be necessary to run the game. This compression while reducing the size of files also increases the load time when the files are necessary and increases memory usage while being decompressed as well. Accomplishing a balance of reducing file size while keeping memory low can be accomplished by using efficient coding that deletes unused variables or other data storage means. This means that maybe some of the most often used images are kept in a smaller library that isn't always loaded. Cloud storage is probably the best way to expand quickly for everyday storage, while hard storage is best for backups and security. Using cloud computing allows the company to scale bigger or smaller as necessary with less overhead and resources to begin with. The ability to scale with storage requirements allows you to upscale if there are many saved games or more high-definition images or videos need to be added to the program. It also allows for a download package that will install the application for clients as well. Down scaling can be helpful if the application is starting out and has yet to be spread amongst users effectively, while still keeping costs lower by not requiring the physical resources to be purchased upfront.
4. **Memory Management**: Types of high memory in terms of Draw It or Lose It include having multiple users connected to a single game for clients or servers. Also, it includes how often you perform tasks like authentication and connection to the server itself. All of these tasks interact together and increase memory usage. It also sounds like there has to be some memory usage for the actual drawing being done for the game itself that needs to be as seamless as possible in order to keep the game fair. This also requires memory resources. Then you also need to worry about whether the client computing device has the memory available to use in the first place. Coding also has a huge impact on memory usage. You can write code that does not clear up memory effectively. Using too many global variables when not necessary or not clearing data after use uses up valuable memory resources as well. Anything that can be stored for long periods of time can be transferred to storage. Linux has a very low requirement for memory especially server side based on the lack of a GUI being necessary to use it. Linux can use Bash to perform tasks that other platforms require multiple menus to get through. While there are ways to do this on other platforms, Linux was created with this in mind. Mobile Architecture requires the application to use less memory as mobile devices have less memory available. Without proper memory management the application will not run as effectively.
5. **Distributed Systems and Networks**: Using the id of the players teams and games makes reconnection easier as the instance can be refreshed quickly and get the players back relatively quickly. The networking requirements might increase over time and require more bandwidth for uploading and downloading data. Keeping these requirements as efficient as possible will help to allow more users to play. It also allows connection between the server and the client. This is the most important facet of networking, though not the only important networking issue to manage.
6. **Security**: Using the private variables and private arrays as the only identifiers with private information, helps data to not be released using other calls. Along with using a REST API of the client-server style we can ensure that roles are allocated and authenticated between users. This allows administrators to be able to manage the back end of the server while still allowing the game to be played without issues. This can help prevent malicious attacks against the program itself and prevent the loss of gathered personal information.