

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 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 | 03/18/23 | Will Key | Added executive summary, requirements, design constraints, domain model, evaluation, and recommendations |
| 1.1 | 03/19/23 | Will Key | Minor corrections |
| 2.0 | 04/02/23 | Will Key | Revised platforms evaluation |
| 3.0 | 04/15/23 | Will Key | Revised Recommendations section and platforms evaluation |

**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 client, The Gaming Room, has approached us requesting help developing a web-based version of their Android game, Draw It or Lose It, a team-based competitive game in which players of each team guess what is being drawn on the screen. The drawings will come from a library of stock drawings. There will be multiple rounds per game, multiple teams consisting of multiple players, and time limits on each round.

The client wants help setting up the environment for the web-based version of the game, and they want us to manage the process of streamlining the design and production of the application.

We will create a Linux-based game server for the client to host their game and distribute it to millions of users for multiple platforms, mainly including Android phones and iPhones.

## Requirements

* One or more teams
* Each team can have one or more players
* Unique game, team, and player names, so players can check whether a name is in use already
* Only one instance of the game can exist in memory at a time
* The game must be in the form of a web-app that can serve multiple platforms

## [Design Constraints](#_2et92p0)

* The game service must be hosted on a web server. This means The Gaming Room will have to host a server themselves or we will have to outsource the server management to a third party company, like Linode, Amazon Web Services, or Microsoft Azure.
* The game environment is made up of games, which are made up of teams, which are made up of players. There is a serious need for organization in the code for which an object-oriented language would be a good fit (i.e. Java, C++, Python). Java would be a good choice because of its platform independence, automatic memory management, and static typing, making it a great language for large projects with many developers that should work on many machines.
* The app must be ported from the existing Android version. This is another reason Java would be a good choice. It is likely that the existing codebase for the Android app is written in Java. So, some of the code may be preserved.

## [Domain Model](#_8h2ehzxfam4o)

The heart of the program is the Entity class. It is inherited by the Game, Team, and Player classes, and it holds the basic properties and functions which these three classes have in common. The default constructor of the Entity class is private, meaning that any Entity class, including its three subclasses, must be constructed by the overloaded constructor which takes a long and a String as parameters.

The Player class is the first subclass of the Entity class. It has all the qualities of the Entity class as well as its own overloaded constructor and its own toString() method.

The second Entity subclass is the Team class. This class is just like the Player class, except it can create unique instances of the Player class with the addPlayer() method, and store them in a List.

The third subclass of the Entity class is the Game class. This class is just like the Team class, except instead of being responsible for creating and managing Player objects, it creates and manages Team objects. So, the Game class manages the Team class, which manages the Player class. All of these classes are subclasses of the Entity class.

The Game class, which manages the Team class and, in turn, the Player class, is managed by the GameService class. The GameService class is a singleton, which means there can only be one instance of it in the whole program at a time. This single GameService instance creates and manages unique instances of the Game class. It also has static methods and variables for aiding in the management of all three Entity subclasses, which keep track of each instance’s unique id and name.

All of this activity is overseen by the ProgramDriver class, which just uses the single GameService instance to create some Games and then manages those Games and their Teams and Players.

The SingletonTester class just executes some code to test if there really is only one GameService instance in the program.

"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** | Mac has a high quality ssh tool and other dedicated server features. However, hardware and software are not customizable. Mac OS is also not free. Does not have access to all the Linux server tools | Linux is free. Linux has high quality sysadmin tools. It is also very flexible and adaptable for any use case. Linux is secure. There are many affordable options for Linux hosting, like Linode. | Not free like Linux. Has an easy to use GUI. Access to Azure cloud. More prone to hacking than Linux. Businesses must pay a licensing fee to host windows servers. Does not have access to all the Linux server tools. | Mobile devices are not a realistic option when it comes to hosting a server. Servers need massive storage and powerful hardware. There is also not much server hosting software written for mobile platforms. |
| **Client Side** | Mac has smaller market share than Windows, but larger than Linux. Will need developers to test the web-app on Mac OS. | Linux has a very small market share. Not worth the investment. But, if we make the app open-source, users could port it to Linux. | Second biggest audience to mobile. Will need to ensure that web-app is optimized for Windows users. | This will be the primary user base. Support for this is not optional. Will need mobile developers and mobile client apps for web-based game. |
| **Development Tools** | Some languages like C++ would have to be compiled explicitly for the Mac platform. It might be more efficient to use a platform independent language like Java. | Many languages can be used to write Linux programs, like C, Python, Rust, and Java. Will need to use a platform independent language like Java. This way we can write one program and port it to all platforms. Might not be worth it to have to write code specifically for Linux. | Windows has visual studio and the .NET framework. Languages include C# and Visual Basic. Will need Windows machines for testing. Most languages and their libraries can run on Windows. Writing the desktop app in a language like Java will minimize need for developers. | Will need Android and iPhone app. Need Swift, Objective-C, Java, and Kotlin developers. Will need Android Studio, which is free, and Apple Xcode, which is not free. |

## 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**: The web-based game should be hosted on Linux. Linux is free, secure, and easy to develop for. Linux is stable, especially if we use a distribution like Debian or Ubuntu Server. Linux is also a very lightweight operating system. The kernel can be incredibly small, and the system can be optimized to run using less resources, which is great for massive, resource-heavy servers. This will reduce the number of developers and hours needed to keep the game up and running. With Linux, we can also avoid the pains of licensing fees. There are also many existing third party server hosting providers that run Linux. Linux also has advanced copy on write filesystems like ZFS and Btrfs which can protect data. Linux also has advanced system admin tools like ssh, which make server administration much easier.
2. **Operating Systems Architectures**: A Linux operating system is built on the Linux kernel. Linux is highly portable, meaning it can run on a vast array of machines. The kernel is monolithic, meaning all the modules and core functionality exist in the same space. The kernel interacts with the hardware to allow user interaction and feedback. The shell acts as a user interface layer and allows different programs to run on the operating system. A typical Linux operating system is split into the user space and the kernel space. All the user’s applications and some libraries exist in the user space. The kernel exists in the kernel space. These two layers interact to allow programs to run using the machine’s hardware.
3. **Storage Management**: The Linux kernel provides a file system abstraction layer called the Virtual File System. This allows the kernel to interact with a wide range of filesystems, such as ext4, Btrfs, and ZFS. It also allows the system to interact with the physical data storage devices in the machine in a standardized way. We should use the ZFS filesystem. It is perfect for Linux servers because it can work with massive amounts of storage, it can handle RAID, which manages multiple block storage devices for mass storage and redundancy, and it is a copy-on-write filesystem, which protects data by allowing the system administrator to save states of the filesystem and revert to old versions. This could help to ensure that the server is quickly back up and running in the event of a mistake being made.
4. **Memory Management**: Linux has a sophisticated virtual memory management capabilities. It can even manage memory for supercomputers, so it will work great for our game server. It has systems for translating virtual memory to physical memory and can manage many processes in kernel space as well as in user space, where our game will reside. The Linux kernel divides memory into 4KB buffers called pages. It tracks the usage of these pages and adjusts the system’s memory to accommodate changes in the amount of memory being used at any given time. Linux also uses a feature called Swap, in which the Linux system transfers some of the pages from memory to the hard drive. Memory is significantly slower to access from swap, but using swap allows the system to lighten the load on RAM in cases of heavy usage. This feature could prove quite useful for something as memory intensive as a server.
5. **Distributed Systems and Networks**: The game server needs to be connected to a massive network with a large amount of bandwidth and high speed to be able to serve the game to millions of users. There also needs to be many servers for matchmaking between groups of players. The client app needs to be distributed for multiple platforms which all need to be able to interact with one another through the servers. Platforms will likely include x86\_64 and x86 for Linux, x86\_64 for Windows, arm64 for apple computers, and Android and iPhone. The server should use the REST architecture and be interactive through the use of HTTP requests. That way the players can make actions in the game and receive feedback from the server, and the other players can see the results of their actions as well.
6. **Security**: Linux is arguably one of the most secure platforms, because it is open-source, meaning anyone can audit the code and make security updates at all times. Linux is also not generally targeted by certain hacker attacks, because not many consumer PCs run Linux. Linux also has a hardened kernel called SELinux (Security Enhanced Linux). Linux also has a sophisticated user privilege model, which restricts root access to certain systems from certain users. This gives the system admin complete control over every process running on their system.

Sources Cited

Tucakov, Dejan. “Linux vs. Microsoft Windows Servers, the Ultimate Showdown.” *PhoenixNAP Blog*, 5 Dec. 2022, https://phoenixnap.com/blog/linux-vs-microsoft-windows-servers.

“Use Mac Mini as a Server.” *Apple Support*, <https://support.apple.com/guide/mac-mini/use-mac-> mini-as-a-server-apd05a94454f/mac.

“Anatomy of the Linux Kernel.” *IBM Developer*, https://developer.ibm.com/articles/l-linux-kernel/.