

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 | 5/25/24 | Robert DiMaio | Updated Executive Summary, Design Constraints, and Domain Model. |
| 2.0 | 6/8/24 | Robert DiMaio | Updated Evaluation table. |
| 3.0 | 6/22/24 | Robert DiMaio | 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)

The Gaming Room has created a game “Draw It or Lose It” that is currently only available as an Android app. They have reached out to us as consultants in order to translate their gaming app into a web-based version that serves multiple platforms. We will be utilizing various design principles and object-oriented programming (OOP) principles to create this for them using Java. What we will need from them is a budget and timeline requirement in order to get a better estimate of how much work and how many people this will take. The software requirements are listed below in the next section. Once we start making software application decisions, we will then also need to establish hardware requirements.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

Some of the design constraints for developing in a web-based distributed environment are which languages/IDEs to use, reliance on servers and hosting, and the user’s internet speed. We are currently developing in Java, likely because that is a common language in Android development, but it may not be the most appropriate language for developing something web-based. We must make some decisions on how we are going to be hosting the game, what servers we are using, etc. We also have to take into consideration a user’s internet speed, or even access to internet in the first place. If a user had the app downloaded on their Android device, they may have been able to play the game offline. This will not be possible if it is web-based. We may also take for granted our stable internet speeds, but some parts of the country, or the world, this is not always the case. We must plan for that as well.

## [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 UML class diagram below shows the relationships between the classes of our program. The Game, Team, and Player classes all inherit from the Entity class, which naturally exemplifies the OOP principle of inheritance. Each of the child classes have the same attributes id and name, and methods for the default constructor, getId, getName, and toString. Each of them can override the parent class’ constructor and toString methods, which is an example of polymorphism. Polymorphism allows for child classes to implement inherited properties depending on which class is calling the method. The private attributes of each class is an example of encapsulation, which hides an object’s internal functionality, preventing unauthorized data access. The fact that the program is separate into classes also shows abstraction, because we may not need to know how some of the classes work “under the hood” so to speak, as long as we can utilize them appropriately. The diagram also tells us that there is a “zero-to-many” relationship between GameService and Game, Game and Team, and Team and 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)

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** | MacOS is based on Unix, which offers a stable and secure environment. It is typically used for development rather than server hosting, however. Some of the advantages are its security features, stable Unix-based architecture, and ease of compatibility and integration with other development tools, products, and services, especially other Apple-based elements. The weaknesses are that Mac has generally more expensive hardware than comparable Linux or Windows machines, as well as limited support due to a smaller community and fewer resources for server side hosting as compared to a standard such as Linux. | Linux is open source and highly customizable. It is the most popular choice for server side hosting. Its advantages include its cost, as it is often free, high stability and performance, strong security features, and a vast pool of knowledge and resources for support due to its large community. Its weaknesses are its steeper learning curve for developers due its reliance on command line interfaces, some compatibility issues as compared to Mac or Windows, and it requires more hands-on maintenance. | Windows is a well-known and often-used OS with a user-friendly interface. It has great compatibility with a huge amount of different software and services. Some of its advantages are its familiar and easy-to-use GUI, strong support and integration, especially for Microsoft tools and frameworks, and comprehensive support available, both commercially and via other users. Its weaknesses include high licensing costs, especially compared to Linux, which can be free, higher resource usage compared to Linux, and potentially less secure than Unix-based counterparts. | Mobile devices (most commonly iOS and Android devices) have limited resources as compared to desktop-based server environments, and are typically used for app hosting rather than traditional web servers. Some of their advantages are their ability to host small-scale services, such as IoT (internet of things), direct access to mobile-specific features, and their increasingly powerful hardware as mobile technology advances with modern devices. Despite these, their weaknesses include limited processing power and memory compared to desktop based systems, less flexible development environments, and less reliable network connectivity compared to wired connections of traditional servers. |
| **Client Side** | Using MacOS would entail a higher initial investment due to expensive hardware, though development tools such as Xcode are free. There may also be additional costs for third-party libraries or services. Development time could be quick due to Mac’s integrated environment and robust tools, however, updates and OS changes may require adjustments. The expertise required would include teams that are familiar with Mac-specific APIs and development tools such as Xcode and Swift. Apple’s ecosystem integration and guidelines would also require some time and knowledge. | Linux requires a lower cost than other OSes due to it being open-source and no licensing fees. There may be additional costs for specialized support or distribution. One of Linux’s potential benefits is how configurable it can be, which can be a negative when it comes to time investment, though experienced Linux developers can move quickly. Linux will require command-line and system administration skills for expertise. There are also various Linux-specific tools, environments, and distributions. | Windows requires licensing costs upfront for their own OS, as well as development tools such as Visual Studio. There may also be costs associated with proprietary third-party libraries or frameworks. Windows shines in its time considerations due to user-friendly tools and extensive documentation. Developers will need expertise in Microsoft development environments, security and deployment practices, and Windows-specific technologies. | The costs for mobile devices (iOS and Android) may be higher due to needing specific hardware and development environments for each OS, such as requiring a Mac for developing for iOS. There are also developer program fees for each OS. Development time can vary significantly depending on the complexity of the app. There will be regular updates which can also add to the time required. Developers will need to be experienced in specific languages and tools for mobile development such as Swift for iOS, and Kotlin or Java for Android. Familiarity with each app store guidelines, submission, and distribution processes will also be beneficial. |
| **Development Tools** | MacOS uses the languages Swift, Objective-C, Python, Ruby, and JavaScript. The IDEs and tools necessary would be Xcode, AppCode, Atom, VS Code, and Homebrew. | Linux uses C/C++, Python, Java, Ruby, PHP, JavaScript, and Bash/Shell scripting. Linux would use Eclipse, PyCharm, VS Code, Vim, Emacs, GCC/Clang, Make/CMake, and Docker, to name a few IDEs and tools. | Windows uses C#, C/C++, Visual Basic .NET, JavaScript, and Python. A Windows-specific IDE that is commonly used is Visual Studio (not to be confused with Visual Studio Code). Windows also uses VS (Visual Studio) Code, JetBrains Rider, and IIS. | The main languages used for iOS development are Swift and Objective-C. The primary IDE for iOS development is XCode, which comes with all Macs by default, as well as AppCode and CocoaPods and Carthage. Android uses Kotlin and Java for its main languages. We would use Android Studio, IntelliJ IDEA, Gradle, and Android SDK for its IDEs and 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**: The operating platform I would recommend for the server would be Linux. Linux is the most popular server-side hosting operating platform for several reasons. Depending on its usage, it can be free as opposed to other OSes which may be costly. It has high stability and performance, as well as strong security features. Due to its widespread use, there is an endless supply of support and knowledge available online from its community of users. We will be most interested in its stability and reliability, and there is no need to “reinvent the wheel” and try to utilize a different OS that is not as well-suited for our application. Using Linux for our servers will be our best option to allow The Gaming Room to expand Draw It or Lose It to other computing environments.
2. **Operating Systems Architectures**: The Linux architectures include the hardware considerations, such as processor, storage, network interfaces, and memory. The most common processor architecture is x86\_64 (AMD64), but ARM has become increasingly popular due to its power efficiency. Ethernet is the most common network interface used for a physical Linux server. Storage and memory is discussed below. There are many OS and kernel configurations available that are beyond the scope of this paper, but the distribution Ubuntu Server would be a good choice for what we need. Security details are also discussed further below.
3. **Storage Management**: Hard-disk drives (HDDs) and solid-state drives (SSDs) are the standard solution for storage management to be used with Linux for servers. SSDs are much faster than HDDs but are more costly and generally come in smaller capacities. HDDs are a great solution when we need high-capacity storage that will be reliable, but for our purposes, SSDs will suit our needs best. Given our storage needs, which include 200 high resolution images that are 8 mb each, plus the text files and other assets that will need to be utilized by the game, the sizes available in SSD format should be more than enough. If we had to store many more or much larger files, it may make sense to use HDDs, but given what we need, the increased speed and reliability of SSDs make them worth the extra cost. SSDs are more reliable and last longer than HDDs, which often need to be replaced within 5 years due to physical moving parts that SSDs do not utilize.
4. **Memory Management**: Linux uses many sophisticated memory management techniques including virtual memory (paging and swapping), memory allocation, page caching, and shared memory. These all work in tandem to make the most of what physical memory we may have available and maximize resources. Draw It or Lose It will need to run 1000 concurrent games, with 4 players each, meaning it needs to be able to handle 4000 clients at a time. Each of the 1000 games will be using a randomly pulled 8 mb image, for a total of 8 gb of memory, plus more for the other tasks of running the game. Random access memory (RAM) is what will be used for memory management. The two main types are dynamic random-access memory (DRAM) and static random-access memory (SRAM). SRAM is the fastest memory available currently but is also the most expensive. To efficiently handle this many clients reliably, the cost of using SRAM is well worth it to give the players as smooth and reliable of an experience as possible. We should be utilizing as large a capacity of SRAM as possible that fits within The Gaming Room’s budget.
5. **Distributed Systems and Networks**: Some of the key features of distributed systems and networks are decentralization, scalability, and fault tolerance and reliability. Decentralization means there is no single point of failure and resources are distributed across multiple servers. This decreases workload on any single server as well as increases redundancy and replication, leading to better fault tolerance and reliability. If a single machine fails, it will not bring the entire application down because another machine can pick up the slack and the workload is distributed amongst the rest. Utilizing distributed systems also lends itself to increased scalability, because we can always add more servers to increase performance to handle a larger workload. All the clients and servers will need to be connected via a network. The distributed nature of the network means that if a single server fails, there will be others that can handle the task. This redundancy prevents outages from being as catastrophic as they could be with any single point of failure, such as with just one large server connected to all the clients.
6. **Security**: Some security feature available include security keys, usernames and passwords, biometrics, and multi-factor authentication. These would all be present client-side and would aid in authenticating the users to protect them and their information, as well as protecting our application from unauthorized users. These have all become standard practices for any program or application that users may access with their individual accounts. Some of the server-side security features present in Linux are firewalls, authentication and authorization, and monitoring and logging. Linux allows us to set different permissions depending on the type of user (such as a normal user vs an admin). Encryption is also standard practice for keeping data safe. For example, there is a tool called Linux Unified Key Setup (LUKS) that protects the data that stays on the storage disk, keeping users’ information safe.