

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 | 03/16/2023 | Kaylee Marting | This is the first inspection of Draw It or Lose it. |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose it is an Android based team game that is to be ported over to web. In this game, users are to guess a drawing within 60 seconds. If they do not guess correctly, one of the opposing teams have 15 seconds to guess what the same drawing is. Given that this is a team-based game, the game must have team functionality. Specifically, Draw It or Lose It will need to have the ability to create teams, and assign users to teams within that game. Each team will require a unique name, so it is imperative that the program is able to search the team names to ensure the name is not currently in use. This will be relatively simplified by the program only allowing one game to be running in any instance. Lastly, the program will run on all operating systems, so Java will be the language utilized as it is able to run on all systems.

## Requirements

To create a streamlined experience for users, it will be required to port the entirety of Draw It or Lose it over to web. This will open accessibility to iOS and computer users, as the game is currently only available on Android. Due to the team playing nature of the game, the program will require multiple teams with several users per team. Each team must have a unique team name that is currently not in use, and the program will not allow multiple instances of the game to run at any given time.

## [Design Constraints](#_2et92p0)

Draw It or Lose it will need to run on all operating systems to open availability to everyone. This will require using a language that will run on all systems – Java is the suggestion at this time.

Regarding iOS users, Swift is the required language to build iOS apps so if it is decided that this game will be ported to the App Store, it will need to be rewritten in Swift. This will not impact the web launch, as iOS users will have access via mobile web.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

ProgramDriver and SingetonTester utilize each other to function – ProgramDriver houses the main() function that is public, and therefore accessible to the rest of the program, and SingeltonTester tests that the program utilizes to ensure there is only one instance of the game running at any given time. Entity stands as a super class, with four classes that inherit from it. Inheritance values include the name and id, as well as the functions to get the name and id, as well as toString(). GameService is the first class to inherit from Entity, and this is the class that drives the functionality of the game creation. It houses the attributes games, nextGameId, nextPlayerId, nextTeamId and service, all of which will be private. Opperations include GameServices which is private, and getInstance, addGame, three instances of getGame that will pull the id, name and count, as well as getNext which will pull in the user name and team information. This will allow zero to many instances and works along side Game, Team and Player classes. Game will take care of the teams list, which is private, as well as the ability to create a game and add teams and will set the names of each. Team handles the players list, as well as the ability to add players names into the game. Player handles the player ID strings, allowing the player information to be stored.

Each of these classes are associated with one another, and all directly inherit from the super class Entity. Game and GameService are connected via association, as is Game to GameService and Team, and Team to Game and Player. This signifies that while each class carries out its own functions, they are interdependent of one another. The game will not function correctly without Team and GameService carrying out its orders and so on.

**"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** | Given the usability of the terminal, it will be relatively simple to run commands and better track the program’s functionality. It’s a very secure system that will provide protection against malware etc. macOS servers tend to cost around 1,000 for an unlimited client server, and do not need to be updated to the next tier when a new release is out. This unlimited client server would allow the game to grow in popularity without crashing if too many users were online at one time. | Linux has the same power and functionality in the terminal as macOS but has the added advantage of being a free open-source operating system, making it more accessible to potential users. This can allow setting the server up with unique requirements a bit easier, as there is more control allotted in the Linux environment. Yearly Linux servers can run between 250 and 1300 dollars a year, depending on the option. Given they are not purchased outright, this option is not recommended, the overall cost would add up quickly. | Given this is the most popular operating system, it will be easily accessed by a larger group of users, thus eliminating concerns about cross compatibility right at the beginning. Users would not require a large learning curve to get in and start developing the product and would be able to easily implement a cloud-based server system on Windows. Leasing a server would be between 25 and 125 dollars a month, depending on which option was chosen. Purchasing would cost between 1 – 6 thousand dollars. The server would need to run at minimum two years to break even on purchasing. | Utilizing the App and Play stores to launch and maintain the app will be the smoothest option. Given the Android app is already implemented, this is not a concern. iOS will require Swift and a computer running macOS. It can cost between 250 and 500 a month to keep the app up and running on the server, but the costs will go down as the users flood in, bringing an influx of revenue. |
| **Client Side** | Moderate user comfort will be required to set up Mac. Unless the user is already utilizing macOS, it will not be necessary to switch. It would be mandatory to host a server that can easily handle the backend requirements and requests of the user, and to ensure that Safari as well as all the other browser options worked. It would be worth going so far as to ensure arm and intel chips functioned, and to even install edge onto a macOS device to test it out. iPads and iPhones should also be tested. | Getting the client up and running will be a bit more difficult as Linux requires a wealth of knowledge to utilize. Given the requirements, it’s recommended this system is not used, as it would be cost prohibitive to host via Linux. Instead, it would be best to host via Windows or macOS and ensure Linux users were able to utilize the website without error. | As Windows is the most widely used operating system, getting the program up and running with this will be relatively simple. To ensure all browsers were supported, robust testing across various methods would be mandatory, including side loading Windows onto a macOS machine, in order to ensure all browsers work as expected. | Given Android is the most widely used mobile operating system, there are a multitude of programs available to launch and track Android apps. iOS apps can only be built using Swift, on macOS, thus limiting its usability. This app would have to be tested in tandem with building to find any blocking bugs before they went to production. |
| **Development Tools** | Eclipse would be the IDE of choice to program the Java code. Given it is open source, it will be the best option financially. CLion is another option, though it requires an enterprise account to utilize. Visual Studio Code is another option for macOS, as it is free to download and runs Java. Given VS allows for cross platform development, it will be idea to utilize despite the operating system that is chosen. It is free to use, simplistic to learn and very robust, allowing for extensions and many options for developers. | Eclipse is the go-to option for Linux users, and given it’s compatible with all other operating systems, Eclipse would be a good choice for Linux. Visual Studio Code is another option, as it runs on Linux, and can be modified to create the ideal IDE for users. Given its cross platform compatible, it would be a lot easier to have a larger team using multiple OS to develop code, though mobile devices would have to be handled with Visual Studio, rather than VS Code. | Java runs on Windows without a hitch. Options for IDEs include Eclipse, IntelliJ, NetBeans and so on. Eclipse is the best selection due to the cross-platform compatibility. Visual Studio is another option, as it is a Windows based program, and allows Java as well as various other coding languages. It does not require a subscription or purchase, simply a computer running Windows. | Eclipse is a good option for developing more for the Android app, as it’s open source and the initial work is being carried out in Java. For iOS users, the program would have to be refactored in Swift, but again, this is a free to Mac program. Visual Studio would be of benefit to mobile development, though it would not handle iOS, as discussed above. Instead, it would be used to handle Android, and it would be ensured that the apps operated the same on Android and iOS. Android Studio is another option as it runs on Windows, Linux and macOS. |

## 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**:

Utilizing Windows and Eclipse to create and launch The Gaming Room’s program will be the easiest and most straightforward option. Windows is a well-known and easy to use system that will be familiar for most. Eclipse is an open-source IDE that functions on all systems, except for iOS apps. Given the intent is to launch a web-based variant of the game, it stands to reason that concern for users on iOS and Android won’t be a concern. That said, ensuring the game functions properly on mobile browsers is a requirement for launch. If Windows were not the only system running in the company, Visual Studio Code would be a viable option, as it operates on Windows, macOS and Linux. This would allow more developers to have access to the code without concerns of IDE implementations mismatching, as well as instant access to the code database that will allow faster access to begin working.

**Operating Systems Architectures**:

Windows utilizes a layered architecture approach. Users will mainly interact with the programs themselves, rather than using the terminal or specialized commands to run the system. The main benefit to this type of architecture system is that the system does not need extra programs running to play Draw It or Lose It. The program will be able to run and function on its own accord. This allows for a user-friendly experience wherein the players will need no specialized expertise to access the game. This also applies to macOS as it is a web-based game. For iOS and Android, running on the web browser would need to be fully tested to ensure that mobile view works as expected. If an app were to be built and uploaded to the App or Play stores, it would require building the app in the correct language, Java for Android, Swift for iOS. The game running on Xbox or PlayStation would also require either allowing it to be played via the server on web, and ensuring the controls all worked as expected, or creating a game that can be downloaded onto each system.

1. **Storage Management**:

Given the game will only allow one instance of play at a time, memory will not be a massive concern, however, to keep with best practices, the game should not take a large amount of space. Utilizing cloud-based storage will be the best option for the game, as it will be instantly accessible for all users, regardless of the platform of choice. If Cloud Based storage is not an option, creating an app that can sync with the servers would be an acceptable choice.

1. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>

Storing the game in a centralized location and allowing users to access it via the cloud will ensure the game is stable, protected and cannot be deleted via an attack. That said, Java is excellent at managing memory due to automatically cleaning up unused data. This will limit the need for storage on the cloud and help to keep the game running smoothly. If a system fails, it should be noted that the game must fail gracefully – in that sense, the game should not crash or otherwise harm the operating system. Instead, a polite message should be implemented, alerting the user that the system is currently undergoing fixes.

1. **Distributed Systems and Networks**:

Utilizing a client-server system would be the most beneficial, the client would be able to access the game via the cloud, and it would be maintained and protected on The Gaming Room’s servers. As a distributed system, there would be the added benefit of concurrency running. This would allow the game to keep running while one portion of the system went down. Users would not be disrupted while the company worked on trouble shooting the problem at hand. As the game becomes more popular, especially with the addition of multiple client types, it will be beneficial to the company, as they can scale the network bandwidth to maintain the increase of users. If the system were to be set up so that each server type handled a specific OS, such as a server for web, a server for Android, a server for PS4 and so on, a potential outage would be sequestered to the specific OS type. This would also allow specific teams to handle each server, rather than a larger team handling one point of potential failure.

1. **Security**:

Given security is of the utmost importance, care should be taken when programming, using the best practices to ensure there are no data leaks, easy routes into the code, or general access for nefarious behaviors. The system should be regularly scanned for viruses, and a firewall would be a good option to protect against attacks. It would be beneficial to create account password protection down the line, to give users access to their profiles across various platforms. This would add an extra layer of security, as well as drive users to participate in the game more.

Additionally, creating a two-factor authentication system for the program would help ensure further safety measures are implemented – this could be as simple as an email or text with a code that the user must input into a pop up that shows they are in fact the intended user. Conversely, requiring email verification prior to the game being accessible would limit bot attacks as it can be challenging to automate checking a fake email to crack into a site.