

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs) **1**

[**Table of Contents**](#_heading=h.30j0zll) **2**

[**Document Revision History**](#_heading=h.3znysh7) **2**

[**Executive Summary**](#_heading=h.2et92p0) **3**

[**Design Constraints**](#_heading=h.tyjcwt) **3**

[**System Architecture View**](#_heading=h.3dy6vkm) **3**

[**Domain Model**](#_heading=h.1t3h5sf) **3**

[**Evaluation**](#_heading=h.vb1vn2hxrq4k) **5**

[**Recommendations**](#_heading=h.3rdcrjn) **10**

**References 12**

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/29/2022 | Kyle Wucik | Initial document add notes for design: Executive Summary, Design Constraints, and Domain Model |
| 1.1 | 06/5/2022 | Kyle Wucik | Updating Document to include notes for server architecture: Server Side, Client Side, and Development Tools |
| 1.3 | 06/15/2022 | Kyle Wucik | Added final recommendations for how to run the game’s server |

## [Executive Summary](#_heading=h.2et92p0)

Current client, The Gaming Room, wants to turn their android based game (Draw it or Lose it) into a web based one. They want the web based game to be supported through multiple platforms, such as MacOs, Linux, and Windows. There are four rounds that last around a minute each and it has players guessing what is currently being drawn on screen.

## [Design Constraints](#_heading=h.tyjcwt)

A few design constraints come to mind when designing a game like this:

* **UI:** The user interface is currently on an Android ecosystem and needs to be formatted correctly to be used on multiple web based clients. To make this work we will need to take the current design from Android phones and make it more spread out to fit wider screens. Another option is that we could redesign the game for the web or for mobile devices. It could look like a new game altogether or it could just improve upon what is already there. We don’t want the web based app to look too different or have features the Android version does not (or vice versa).
* **Potential Copyright Claims:** Depending on what is being drawn and how it is being drawn we might need to get permission for certain images. The Android app, hopefully, has a handful of images that they are legally allowed to use already, so we can start by importing those ones and get more as time passes. This will allow players to play right away, even if we don’t get the rights as soon as we want, and this will provide new images in a way that makes the app feel fresh with every update of the image section.
* **Cross-Platform Coding:** The Android app’s code might not work well with the web-based version of the game. I think it’s important to be mindful and understand that there could be issues with the code and cross-platform requirements of this project. Until the actual code is built there is no way of knowing if there will be issues. It might be beneficial to recode the Android app completely and make sure it is compatible with the web-based version. This will prevent the issues that we run into and it’s better to prevent them than have to fix a bunch of stuff that nobody saw coming.
* **Memory Management:** Only one game will exist in the code’s memory at one instance. There is also the fact that every player, game, and team will have their own IDs. This will need to be looked at when managing the memory of the game and we will have to find the best approach to handling this through multiple platforms.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

The Diagram below shows how the code is to be set up. First of all there are 7 classes that have their own code and attributes to them. We have ProgramDriver, SingletonTester, GameService, Game, Team, Player, and Entity. The GameService class is the meat and potatoes and does a lot of heavy lifting. It is what holds most of the game’s code and is what holds the functionality. Entity holds variables and attributes that would be used in multiple classes. This helps with redundancy and makes the code look neater. As you can see there are three lines going into the Entity class from Game, Teams, and Player the lines have an unfilled arrow at the end indicating that they inherit information from the Entity class. GameService, Game, Team, and Player are connected by various lines with numbers in them (0…\*). This means 0 to many or 0 to more. It shows that every object in the previous class is associated with 0 or more objects in the class that it is connected to. Finally, the ProgramDriver and SingletonTester classes show that the SingletonTester is used by ProgramDriver to test the code and see if there is one instance of the game in the memory, as this is a current requirement.

**"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](#_heading=h.2s8eyo1)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * MacOS was designed by Apple and can be limited in what it allows the user to do. * However, Apple has a really good GUI (or had they discontinued it this year) that allows a user to manage a server right from their OS. * They outline some specific features that the program has on their website “...Caching Server, File Sharing Server, and Time Machine Server...” * While buying a Mac computer that can run a server for a long time without hiccups might be expensive, it's important to note that macOS includes (or included) the server program in many of their OS versions. “...are bundled with every installation of macOS High Sierra and later, so that even more customers have access to these essential services at no extra cost” * Using Mac’s server program will be more beneficial to those that are using a Mac computer, as it was designed with them in mind. | * Linux is a popular open source program that many companies use already for their server hosting. * There are already a variety of different applications that one can use for a Linux based server including Ubuntu and Red Hat. * Because Linux is open source companies will be able to use it at little to no cost and they will be able to better customize their experiences within their servers (especially the security part). * Linux is compatible with almost every programming language, like Python, Ruby, or Perl. * Linux can be very hard to learn, depending on the user’s skill level, but is almost always worth it due to its powerful skeleton of an OS. | * Windows OS is a fairly easy to use and cheap resource. It is also one of the most used operating systems so there are a lot of compatible applications that go alongside working with it. However, they can be costly as many require either a one time large sum of money or have a monthly charge to use the software. * A way around the costly charges is to do a lot of research and know what you are buying into before you buy it (or you can figure out some way to code your own server and just keep up with the coding and maintenance from your end, granted that’s probably what you’ll be doing anyways). | * Hosting servers from a mobile device is definitely not something that is recommended, even though it is possible. * Most of the time you will need to be connected to some sort of internet connection. * Oracle has some software that can be used on Android mobile devices. * However, it is always best to have a stationary server to avoid connectivity issues like constant buffering, inconsistent load times, complete server outages, etc. |
| **Client Side** | * For MacOS, as with most of Apple’s software, there is very minimal time and expertise needed to use any of their software or hardware. * I feel like this would be the most expensive out of the four choices. | * Linux requires a lot of expertise and can be very time consuming. However, the benefit to using Linux is its very low cost, most of the software with Linux is free and the ones that are paid for are **usually** cheaper. | * Windows is kind of a happy middle ground between MacOS and Linux. It’s not too cheap, not too expensive. There isn’t too much to learn, but there are some things that can be very difficult to learn. It’s kind of like a software version of Goldilocks. | * Mobile devices are easy to use when wanting to see updates or other useful information from a user or developer’s standpoint. The best part is that it can be viewed from anywhere. However, there might be some more difficulties when implementing this as the screens are usually touch and there isn’t a lot of support for mobile designs (depending on what is needed). |
| **Development Tools** | * MacOS uses the same programming languages and tools as the other operating systems. * Popular Languages: Typical web page languages such as HTML, CSS, and JavaScript. They are complemented by most other languages like Python, Ruby, Perl, Racket, Java, etc * Tools: PyCharm, Eclipse, Visual Studio Code, in browser IDEs, GitHub, NotePad, etc | * Linux uses the same programming languages and tools as the other operating systems. * Popular Languages: Typical web page languages such as HTML, CSS, and JavaScript. They are complemented by most other languages like Python, Ruby, Perl, Racket, Java, etc * Tools: PyCharm, Eclipse, Visual Studio Code, in browser IDEs, GitHub, NotePad, etc | * Windows OS uses the same programming languages and tools as the other operating systems. * Popular Languages: Typical web page languages such as HTML, CSS, and JavaScript. They are complemented by most other languages like Python, Ruby, Perl, Racket, Java, etc * Tools: PyCharm, Eclipse, Visual Studio Code, in browser IDEs, GitHub, NotePad, etc | * Mobile devices can use the same programming languages and tools as the other operating systems. * Popular Languages: Typical web page languages such as HTML, CSS, and JavaScript. They are complemented by most other languages like Python, Ruby, Perl, Racket, Java, etc * Tools: PyCharm, Eclipse, Visual Studio Code, in browser IDEs, GitHub, NotePad, etc |

## Recommendations

1. **Operating Platform**: Looking over the details of this document I think that the best operating system to use would be Linux. Linux is open source and can be used at little to no cost, which significantly lowers the cost of everything. An effective way to do this would be to use a physical serverless design. Pay as the needs of the game grow and save money on physical server maintenance.
2. **Operating Systems Architectures**: Linux has an architecture with layers of different software. There are three layers of architecture that Linux uses: Hardware, Kernel, and Shell. The hardware aspect is the memory, cpu, storage, and other physical components. The Kernel layer is a translator/communicator with the hardware layer and deals with the hardware/software scheduling. The Kernel is also directly added to the memory to start up the system, during boot. The Shell is the user interface that allows the user to interact with the Linux operating system. It communicates to the Kernel, which then communicates to the hardware and vice versa.
3. **Storage Management**: I think that the best storage management would be to use a server based set up. Doing so will allow for better control of how much storage is needed. From research there are a few good server hosting services that would be good to consider. For budget servers, possibly in the beginning of this application’s journey, I would recommend HostWinds. As time goes on there are better “Overall for Cloud Hosting” (www.websitebuilderexpert.com) services that can be recommended, such as A2 Hosting or Cloudways.
4. **Memory Management**: Linux is a very versatile operating system and most languages work with it. Currently the company is using Java as their code of choice, as seen from the code given previously. Fortunately, “Java uses an automatic memory management system called a garbage collector…” (JavaPoint). Therefore, “...we are not required to implement memory management logic in our application…” (JavaPoint).
5. **Distributed Systems and Networks**: I’m not sure there would be much of an incident here. There are multiple operating platforms that support various kinds of browsers, both similar and different. Being as this game is web-based, I believe that there isn’t much concern about lack of communication. As long as a user has access to an online browser that is kept up to date then they will always be able to access the server and game. However, to make sure there are no issues with outages or possible communication related blips, then a strong server would be needed. A server that keeps up to date with security, storage, and is easily manageable by the client.
6. **Security**: Based on what the client eventually decides to do with their servers, operating systems, and storage space is how to decide what security features we need and what vulnerabilities to look out for. Given that I recommended Linux as an operating platform, I’ll assume the client did too. To have a safe and secure Linux server service means, for example, enforcing password security, only installing necessary packages, and/or having a secure SSH. Password security is important to make sure that others don’t have access to information or options that they aren’t trusted with. Installing only what’s necessary will reduce the risk of malware getting into the system, less of a ratio (on top of this, installing only what you absolutely trust will also help prevent malware incidents). If someone got into the SSH of this server the hacker would gain total control of the server and would have access to everything, including sensitive documents (another reason to use a server based service).

**References**

Apple Support (Not Dated). ServerOS. Retrieved June 16, 2022, from

https://support.apple.com/en-us/HT208312

Carmichael, C. (2022, April 26). *The 9 best cloud hosting providers 2022: Ranked and reviewed*. Website Builder Expert. Retrieved June 18, 2022, from

https://www.websitebuilderexpert.com/web-hosting/cloud-hosts/

Study Buddy. (2021). *Linux - Features and Architecture [ Easiest Explained ]*. *YouTube*. Retrieved June 19, 2022, from

https://www.youtube.com/watch?v=MkPsqAt1\_OU.

JavaPoint. (Not Dated). Memory Management in Java. JavaPoint. Retrieved June 19, 2022, from

https://www.javatpoint.com/memory-management-in-java#:~:text=In%20Java%2C%20memory%20management%20is,management%20logic%20in%20our%20application.