

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 06/19/2022 | Matt Haggard | Draw It or Lose It design constraints |

**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 wants to develop a web-based game called Draw It or Lose it which is currently only available on Android. Being web-based, the game should be multiplatform allowing for a potentially larger player base, allow for multiple teams of multiple players, have a large stock photo library and each game should be unique. The team is unsure of how to set up the environment.

## [Design Constraints](#_2et92p0)

To be truly multiplatform, the game versions must communicate with one another regardless of which one the player uses. The game is already available for Android, so to develop for iOS and web would require 2 different development environments requiring the team to have knowledge of the two which could require some additional training for the team. To develop for the iOS devices the team will also need access some Mac operating system via physical hardware or virtual environment. The game will need to monitor the number and name of the teams and players to ensure that duplicate game, team, and player names aren’t used.

## [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 ProgramDriver houses the main method. It is associated to the SingletonTester to test for multiple instances of the game service and game names. Game, Team, and Player all have a common parent in the Entity class and inherit its methods and variables. The GameService, Game, Team, and Player classes are all associated with one another. The diagram shows their dependency in that you can’t have a player without a team, a team without a game, or a game without a service.

**"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 server offers huge positives if the user is utilizing more Mac hardware. Being a Unix system, Mac is also very stable. The cost of entry, however, is the highest of the options by a fairly wide margin. | Linux has a tremendous upside for server stability and uptime. It also has plenty of free and open-source options to help with costs. The only real downside is that not all versions of Linux ship with long term support. | Windows has a dedicated server OS and it is very well equipped to host an application like this, but an up-to-date copy is more expensive than a Linux server. | Mobile devices have a few server options, but mobile servers are not really a viable option for a larger profile project like this because of software compatibility, stability, and hardware constraints. |
| **Client Side** | Macs are stable and easy to use. The biggest downside to using a Mac on the client side is the upfront cost, especially for a team. They have good software and hardware compatibility. | Linux is great for development with the exception being platform dependent software. Since this is a web-based game, Linux would be a good choice. Also, Linux has several open source and free software, so the upfront cost could be very low. The downside to Linux is the software compatibility can be a little bit of a deal breaker sometimes. In this case however, Linux supports HTML, so it shouldn’t be an issue. | Windows is user-friendly and has great software compatibility with great support for professional development tools. The cost is higher than Linux, but less than Mac. The only real downside to using Windows is security. That can be circumvented with software to an extent, however. | Android and iOS are two of the most popular operating systems in use currently. Their popularity makes developing applications for them a must. They are easy to use and come in a variety of hardware configurations. That said, the nature of the way users interface with them present some limitations that may not exist elsewhere. |
| **Development Tools** | To develop for native software for Mac OS you need an Apple device. The Swift programming language will be used. There are several other options to develop software on a Mac device however. Visual Studio, Pycharm, Eclipse, Android Studio, among others, are all viable options for development, many of which can be used freely or with minimal cost involved. | Linux supports several IDE’s and programming languages. Notable examples of development environments include Pycharm, Code::Blocks, Eclipse, Netbeans, Android Studio and Visual Studio Code. There are a multitude of open-source options as well to keep costs down. Linux fully supports nearly all of the major programming languages C++, Python, Java, C, HTML, CSS, Javascript and has limited support for platform dependent languages like C#. | Windows has a multitude of development tools. Some of the notable tools include Visual Studio, Eclipse, VS Code, Pycharm, Android Studio and many others. There are certainly open-source tools to cut costs on this platform. Windows supports nearly all of the major languages with the most notable exception being Swift. | Developing applications on mobile devices is, while growing, not a very strong avenue. There are a few development tools on a mobile device such as Apple’s iPad OS that allows users to develop applications for Mac devices, but otherwise the tools function with limited support and aren’t very stable. |

## 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**: I recommend using Ubuntu Server to host Draw It or Lose It because of its stability, security, and uptime. A Linux server can run for significantly longer than a Windows server without a reboot. While Mac OS shares a similar benefit, the cost of the hardware makes it a worse solution than Linux.
2. **Operating Systems Architectures**: The Linux kernel is well known for being stable and secure. The Ubuntu distribution is supported by a large community and is documented very well allowing the developers and users to obtain the support they may need in the future.
3. **Storage Management**: The two major storage management devices are hard disk drives and solid-state drives. While solid-state drives offer speed as a huge upside, I recommend using hard disk drives in this case so that the data can be more easily recovered in the event of a failure.
4. **Memory Management**: Since we are restricting the number of instances per game per team per player, a physical memory setup would be ideal for this application. The application doesn’t have to worry about having multiple games open for the same set of teams etc.
5. **Distributed Systems and Networks**: The core of the software will predominantly run on the server side. Developing the all of the systems to connect to this server using their native OS would ensure that the connection is secure and stable. A bit of optimization or conversion may have to be handled after the fact based on the platform the user is interfacing with.
6. **Security**: Security on a Linux server is pretty achievable with a few best practices such as disabling root access, 2-Factor Authentication, only installing relevant software to prevent vulnerabilities, requiring users to use specific password constraints (i.e., mix or upper and lowercase letters, numbers, and symbols), firewalls, and having an up-to-date system.