

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 07/17/2022 | Tam Phan | Added Entity class as a parent for child classes: Game, Team, Player. Modified methods followed by The Gaming Room UML Diagram |
| 1.1 | 07/31/2022 | Tam Phan | Added evaluation for server side, client side and development tolls |
| 1.2 | 08/14/2022 | Tam Phan | Added recommendations for operating system and its architecture, storage and memory management, distributed system and network with security |

## [Executive Summary](#_sbfa50wo7nsh)

Our Client, The Gaming Room wants to develop a web based game, “Draw It or Lose It”. Currently, the game is only available in Android, so they want to make it to multiple platform such as iOS, Linux, and Windows. The game rules are one team guess the puzzle while the application render the image from a large library of stock drawing in a 30-second time frame. The other team will be able to solve the puzzle in a 15-second time frame if the first team can’t guess it.

## [Design Constraints](#_2et92p0)

Multiplatform for iOS, Linux, Windows OS will require an extensive SDLC phases; especially, with the testing phase toward the end of the project.

Team will be split into 3 different sub-team with the main purpose of focusing on one operating system at a time, but each sub team need to have sufficient knowledge and expertise on a specific OS.

Java Multithreading is going to be the main topic for developing the time frame efficiently, but it will be ambitious for this scale of project.

Images needs to be free of licensing to avoid copyright infringement in the future

## [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)

We have Entity is a superclass, and subclasses consists of Game, Team, Player, while ProgramDriver class is the main that use SingleTester class as test case. The UML diagram shows the zero to many relationships meaning object of GameService may know about many objects of Game. It acts the same with Game to Team, and Team to 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Unix-based operating systems, based on Darwin kernel, costly to maintain server, but extremely safe on security level, but only communicate with other Apple devices | Similar to Mac, Linux is another Unix-based system, with different name of kernel from different OS such as Ubuntu, CentOS, Arch-Linux…, free or cheaper cost of hosting, plus open source, and extremely safe. | Most popular OS, user-friendly, but needs to purchase license to maintain server, moderately safe on security level | IOS and Android are the most common on mobile OS, security level depends on service provider, costly to maintain server |
| **Client Side** | Time and cost to build is the most expensive since Apple is not cheap, but moderate effort in expertise | Linux user needs to be familiar with the Linux environment in order to use the system which is required high effort in expertise but the build is cheaper, moderate time in development | Since it’s user friendly, Windows users require low effort in expertise, but time and cost to build is in moderate level | IOs is the same with Mac, but Android is on moderate level with time, cost and expertise |
| **Development Tools** | RAD Studio (IDE) since they all compatible with Mac, Linux and Windows. Swift is exclusive for Apple, but other language such as C++, Java, or Python can also fit with Mac | RAD Studio (IDE), similar to Mac except for Swift | RAD Studio (IDE), similar to Mac, except for Swift | IOs: Swift (Programming Language, Xcode(IDE)  Android: Android Studio(IDE), Java |

## Recommendations

1. **Operating Platform**:

With multi-platform support, Draw It or Lose It can be operated on Windows, Linux or Mac OS. Since cross-platform apply modern programming technique and utilize advanced technology, it will have an optimal support for this application, hence, optimal performance. Windows is best for inexperienced user, Linux suits for high end user and Mac OS is in between those two.

**Operating Systems Architectures**:

Windows and Mac OS have a simple GUI (graphical user interface), easy to install (Windows only), and supports vast majority of application. Linux require technical knowledge to command and control the file directory or set up kernel’s module. Overall, they are all support 32 bit, 64 bit applications or different architecture depends on each of their compatibility

1. **Storage Management**:

Out of the three OS, Windows have the most straightforward process when it comes to storage management. It has an application (Storage Sense) in the system that auto delete unnecessary files since Draw It or Lose It will store the rendered images into the storage space, and it can accumulate gradually, making the hard disk full to the point it affected the performance. Storage Sense make the OS run smoother by cleaning up junk files.

1. **Memory Management**:

Windows is similar to Mac, utilize virtual memory when necessary meaning fetching data to RAM back and forth. However, Linux has swap memory which function interchangeably between hard disk drive and virtual memory. Overall, they all can handle CPU intensive tasks which is rendering the image for Draw It or Lose It.

1. **Distributed Systems and Networks**:

Utilizing AWS Technology provides a strong and reliable connection between server side and client side so they can communicate smoothly. It can be multiples client connect to the same server, and the server internet connection is distributed by the AWS cloud. The main purpose is to avoid outage as much as possible, but it also supplies the server and clients a secure network.

1. **Security**:

Since Windows has a majority of user base, it’s likely to suffer a cyber threat from third party applications. However, Linux, and Mac OS is harder to break since they have layer of security to protect the root system. But everything is hackable, so secure network is a must have with the server and client to prevent data breaches from malicious intentions