

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 | 07/17/2022 | Ryan Fontenot | Revamp of current Android title for multiple platforms |

**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)

Creative Technology Solutions needs to create a web-based game that will serve multiple platforms based on The Gaming Room’s “Draw It or Lose It” which is currently only on android.

The game will work on multiple platforms and will also be called “Draw It or Lose It”. It will have multi-team multiplayer and will have 4 one-minute rounds per game. The app needs a library of static images for the teams to guess on until the timer runs out. Once it does, the other teams will get 15 seconds to guess the images as well.

## [Design Constraints](#_2et92p0)

* Requires servers for multiplayer groups
* Requires multiple teams of several people per game
* Team and players names cannot be the same
* Requires remake for web and multiple different platforms
* Requires image library to pull from

These design constraints are things that we the developers should keep in mind and disclose to the client The Gaming Room. They are technical aspects that could have severe implications for how the game is played and could have a drastic impact on the overall product in the end. The fact that multiple teams of several people is required to play a match could mean that there are less available games to drop into, or maybe the image library requires a subscription. These are things that need to be discussed before starting development.

## [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)

For starters, SingletonTester is used to testSingleton and ensure that there are no entity overlaps.

The GameService class is broken down to the right and each class “has” the next class referenced. This could be called Aggregation. A (GameService) has a (Game), a (Game) has a (Team), and a (Team) has a (Player). They all reference downward and in the UML reference back to GameService with objects like getGame();, getNextPlayerId();, or getNextTeamId();.

Entity however has more of a “is” relationship with the three Game, Team, and Player. The three are all Entities and will Inherit their Id and names from there.

**"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** | +More popular  +Great Troubleshooting documentation  +Upgradable over time  +Multiple server hosting options  -More expensive  -Less accessible | +Very secure in right programming hands  +More accessible and less expensive  +Well known for great server hosting choices  +Good troubleshooting documents  -Less used overall/less support | +Personally, more comfortable w/ windows  +Most clients choice  +Many options for hosting software  +Affordable  -Too affordable = low quality sometimes | +Portable  +Host can be a player  +Most cost-effective  -Worse security  -Higher chance of disconnect issues  -Host player can leave midgame |
| **Client Side** | More expensive than Linux and windows.  Same level of expertise and time as windows required (moderate) | Less expensive than windows and mac.  Requires a high level of expertise and more time to learn and upkeep. | Less expensive than mac, more expensive than Linux.  Moderate amount of expertise and time required. | Less expensive than mac and windows.  Requires a moderate amount of experience and time to upkeep. |
| **Development Tools** | Swift, HTML, Ruby, Java, JavaScript  (More frequent licensing costs) | C++ Visual Studio, Python, Eclipse, Java, Ruby, CSS, JavaScript (pretty much anything)  (Less licensing costs) | Same as Linux but with better support for software  (Less licensing costs, more than Linux) | Android Studio/Swift Pad for app dev.  HTML, Java, Python or Ruby commonly used. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

(I had Windows for this section but wanted to challenge myself a bit from your announcement post and learned about Linux’s cloud servers and security.)

1. **Operating Platform**:

I recommend Linux to host the Draw it or Lose it servers.

1. **Operating Systems Architectures**:

By running Kubernetes, or K8’s we can allow the program/app to “run on a cluster of separate independent hosts.” (Kubernetes.io) which will allow the hardware to get a hand in managing all the games. This is similar to how some video games handle multiplayer matches. By finding the best host, we can lighten the load on the master server & if something were to happen to the temp host, we can quickly jump in to take over the load for the other players.

1. **Storage Management**:

As with most OS, HHD and SSD are interchangeable options. However, SSD is much faster and safer hardware overall. This will decrease load times & allow for faster memory management. This will help our players load their game and the images to load faster for all OS.

1. **Memory Management**:

Linux, like the other options, has the ability to allocate more or less memory to server upkeep depending on the number of current users. This allows for less memory usage when there is less traffic. Throttling usage will save power and costs for memory management overall.

1. **Distributed Systems and Networks**:

With K8, cloud based operating is viable. Because of this, you have the ability to work on your games code, switch active servers to other servers, and allows for back ups when servers crash. This also means that with Linux K8 you could host everything for client end. Therefore, all OS can have a client access the Linux servers.

1. **Security**:

K8 and most Linux server based systems allow for role based security options. This allows players to only access player functionality, like chat & voting in game. While administrators would be able to access more sensitive information like usernames, bans, total players, or password resets for players. This is the security option I recommend for Draw it or Lose it.