

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 | 07/14/23 | Noah Caruthers | Updated executive summary and 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 proposed game, Draw It or Lose It, will need to have the ability to have a single or multiple teams involved. Each team will need to have the ability for multiple players to be assigned to it. The game and team names will need to be unique, unused names, this means that users will need to be alerted to names already in use. Since only one instance of a game can exist in memory at any given time, unique identifiers will need to be created for each instance of a game, team, and/or player.

## Requirements

* A game will have the ability to have one or more teams involved
* Each team will have multiple players
* Game and team names must be unique
* Only one instance of the game can exist in memory at any given time

## [Design Constraints](#_2et92p0)

* Game and team names need to be unique
  + Users need to be notified when a name is already in use. This means that the user will need to input a name and it will need to be checked to make sure that the name is not already being used by another user.
* Only one instance of the game can exist in memory at any given time
  + Unique identifiers will need to be created for each instance of a game, team, and player so that users are able to access the same game with only one instance existing in the memory.

## [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.>

**"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 is easy to use. It has unique capabilities for hosting games.  Growing market in recent years. Built in virus protection. | Linux has good security.  It is not widely used, and it is hard to find supported software. Good for web hosting because of the previously mentioned security. | Windows is the most used OS.  It has unlimited software available. Hardware and software can be most easily changed. It is also most likely to get a virus. | Allows you to take the game anywhere. Non-normal screen ratios, so there is a need to adapt to this feature. Mobile devices have a limited number of inputs. |
| **Client Side** | Moderate cost, time and expertise required. | Minimum cost.  Requires most time and expertise. | Moderate cost and time, similar to mac. Minimum expertise required due to wide support of languages. | Harder to implement. Cost varies. Easy to update. |
| **Development Tools** | Swift is the most common language used on mac, although it supports more. Visual Studio and code runner are compatible IDEs | Works with Visual Studio and Eclipse. Supports nearly every language. | Can use nearly any IDEs and most languages will work. Combination of  HTML/Java/C++ can provide a basis for most games. | Depending on the OS will need to use swift or JavaScript for most |

## 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 would recommend using a Linux Server such as the Ubuntu Server or CentOS.
2. **Operating Systems Architectures**: This type of operating platform allows for the optimization and capabilities of using the Linux OS for the coding aspect and allows for premium protection for the user’s data as well as the owner’s data.
3. **Storage Management**: Using cloud storage with this operating platform will allow the users to switch between devices and maintain relevant data to their profile. For storage of the game itself local storage should be used to load the game efficiently.
4. **Memory Management**: This operating platform will take advantage of memory caches which allow your computer to store recently used images and other data and load it faster than if it were to search through the cloud for it again. This will reduce the time it takes for images to appear on the screen.
5. **Distributed Systems and Networks**: Since this game is cloud based, it will need to have servers that are not hosted by users. This will prevent network outages from having impacts on the users. If a server does crash, the storage node, preferably a NoSQL node, will be relocated to an empty server and the game can resume from that point. This will allow users to not lose progress on an already started game.
6. **Security**: Using the Linux system reduces the risk of a server-side attack. To prevent user data leaks, I would also recommend password requirements and a role-based security system so that only admins have access to important data.