

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 | 12/10/23 | Trevor Hoxie | Made changes to the rocommendations |

**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 would like a web-based version developed of their game Draw It or Lose It. The finished product should be able to have multiple teams, multiple players on each team, only allow unique names for teams, and must only have one instance of the game existing in memory at any point.

## Requirements

Must be a web-based programming language.

There must be multiple teams with multiple players on each team.

Team and player names must be unique.

There can only be one instance of a game at any given time.

## [Design Constraints](#_2et92p0)

Draw It or Lose It must be written in a web-based programming language in order to fully function in a web-based environment.

Has to be programmed to allow multiple teams with multiple players per team by implementing team and player objects.

The team and player names have to be checked for uniqueness; this can be accomplished with a singleton pattern.

There must only be one instance of the game in the memory at any given time. This can be accomplished by creating a unique ID for each instance and checking it using an iterator pattern.

## [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 Game, Team, and Player classes all inherit form the Entity superclass allowing the attributes and operations contained in the Entity class to be written once and used by each of the classes inheriting from it. The GameService, Game, Team, and Player classes are all associated with each other, and use a zero to many associations so they can use as many instances of the other as they need to, or they can use none. The ProgramDriver class drives the package and inherits/uses the SingletonTester class while doing so.

**"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 hosts the server Mac OS X for little cost to the customer, but Apple is not as popular for performing gaming tasks. | Linux has many different options for web servers that would be low cost and open sourced, but Linux has a learning curve for operation. | Windows has its own server hosting, and it would be easier to find someone to operate a Windows-based server, but the cost will be higher. | Most phones do not have the same hardware capabilities as computers so hosting a server from a phone for thousands of players would not be advisable. |
| **Client Side** | Cost would be higher with Mac since most applications are not open sourced. Time would depend on the person hired to maintain the server, e.g., someone who is familiar with Mac versus someone who is not familiar with Mac. | Cost would be low since most things on Linux are open sourced, however it may be harder to find someone who is familiar with maintaining a Linux-based server. | This would be same as operating on a Mac server, most applications are not open sourced, but cost could go down because of the prevalence of people with Windows experience. | Cost for mobile devices will not be much of an issue since there are a large number of server hosting applications, and experienced people, but it will be outweighed by the number of mobile devices needed to support the server. |
| **Development Tools** | Swift is the most common language for developing on Mac and is very popular for app development. It is not a popular a choice for web development as Linux or Windows, however. | Linux is well-suited for software development due to its open-source nature and the availability of many free and open-source development tools. Linux is particularly popular for web- based development with libraries Ruby and Node.js | Windows is particularly popular for enterprise software development. There are many popular languages compatible such as C# and Visual Basic | Mobile devices are not typically used for software development, although there are some development tools available for mobile app development. Many popular development tools and frameworks, such as React Native and Flutter, support mobile app development for both iOS and Android. |

## 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**: Based on the needs of the customer, I recommend the Windows operating system. It is one of the most popular operating systems which would offset the cost of having to pay someone with Linux experience or Mac experience for maintenance. Ther is also a large number of available IDEs in order to program the game to meet their specific needs.
2. **Operating Systems Architectures**: The Windows architecture allows for applications to utilize the platform's kernel processes without directly affecting those processes. In other words, applications can utilize the power of Windows to have a GUI/window set up, access to memory and other vital processes that make the application without inadvertently affecting the processes that make the operating platform work.
3. **Storage Management**: Windows has an entire tool for storage management known as Disk Management which allows for advanced storage tasks. It also has Disk Cleanup and Storage Sense for the task of cleaning out unnecessary files.
4. **Memory Management**: Windows also has the Memory Management tools built in, but a database would need to be made for the game’s image files so they can be accessed by the application.
5. **Distributed Systems and Networks**: The client-server distributing system will have the most merit for this project since each client application will have to depend on a server for the game. There will also need to be a strong network connection since there will be multiple people playing on the game’s server at any given point.
6. **Security**: Windows has the Windows Defender firewall built in that functions as any type of security that the client could need for their servers. However, packets of data being sent to and from the server would need to be encrypted.