

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 | 05/21/2023 | Tyler Rape | Original Version |

**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 to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It. Currently, this game is only available as an Android app. TGR would like to add the ability to create one or more teams, with multiple players, requiring unique IDs. The software must also contain only one instance of the game in memory at any given time. For this, I have added iterator patters to add Teams and Players. We have additionally added code to check for unique identifiers.

## Requirements

A game will have the ability to have one or more teams involved. Each team will have multiple players assigned to it. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

One or more team needs to be involved. Ability to add multiple players. Unique game and team names that the software will check to see if they are in use or not. Only one instance of the game in 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)

Based on the below UML chart, we have a parent class, known as Entity and three children classes. The three children classes are Game, Team and Player. These classes inherit from Entity and extend it. The Player establishes the player ID. The Team complies the players and adds them to a list. The Game class compiles the lists from Team and Player. Game Service is what gets and holds the information from the Game class. Each of these classes can have one or many based on the

**"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** | Historically considered a niche product. Until recently, it was not seen as a great option for service-side development. It is both open and closed source. Considered more secure than Windows. | Linux being open source makes it cost effective to develop. Well known for hosting software applications. It is considered more secure and flexible than other servers. | One of the largest risks of Windows is it is typically more of a risk for security. Since it is widely used, it is a target of cyber-attacks. Likely the most compatible. | Typically works through open-source software. Can have security issues. |
| **Client Side** | Well known for being great at client-side. Many people now have Apple products. As referenced above it is no longer a niche product and is much more widely available. There could be additional cost and time to setting this up for a macOS as it is only partially open source. Would likely need someone well versed in creating software for Mac. | As stated above, this is a very cost-effective operating system to work with. The only drawback is that it might be the least important software to work with for developing the client side. It is not as used in the client world. | The most widely used software, aside from mobile device OS. This being the case, it will have a low cost to develop client-side software as folks are familiar with creating it. | Very cost effective and easy to develop. The Gaming Room already should have some familiarity with this as they already have developed for Android. Typically considered the most flexible amongst the other contenders in this category. |
| **Development Tools** | Primarily for Mac, Objective-C and Swift are the languages that are needed. Apple has created its own IDE and that is called Xcode. | C is the language that you will need to use to code with Linux. The most common recommended IDE that I found to suggest was Visual Studio Code. Many websites recommended this as their top pick. | Windows is written in C++. Visual Studio is unsurprisingly also the most common recommended IDE. It is made by Windows. | For iOS devices, the language will again be Objective-C and Swift. For Android, it is Java and Kotlin. Android also has Android Studio which is the official IDE created by Google. |

## 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**: Considering that all applications could be made on any device, a Windows based device seems to be the best choice for operating platform. This is because Windows devices are cheaper to acquire, require less training overall, and are the most accessible.
2. **Operating Systems Architectures**: The operating system is made up of the kernel, shell, and applications. Since this is an application, it would work in the latter part of the previous explanation. While the application would act within the other areas, we are primarily focused on it.
3. **Storage Management**: Creating a streamlined application is what is most important. Creating cloud-based storage for the non-necessary components of the application will be key. For functionality in low data situations, having all operating components directly in the application, and enough pictures saved for one round of the game, will be required.
4. **Memory Management**: Since we want to minimize resources used in the memory of the device, only volatile information that is not necessary for the game’s function would be stored in the short-term memory. That means, only the active game that is being played would function within it. This could store temporary large picture files downloaded from the cloud storage.
5. **Distributed Systems and Networks**: Knowing that we want the game to function across platforms, while catering to those who need low data functions, I propose that in addition to the Android application, we create an iOS version, as well as a web-based version. While this could be costly up front, it will provide the application to the largest number of users. The Android and iOS versions could have enough pictures saved to the storage to play one game offline in low data mode. The web-based could be available anytime there is internet available on all other devices with a web browser. This means one program could be made for all devices aside from mobile ones. This mitigates outages on devices and allows for a diverse audience.
6. **Security**: Where data is stored and what data is stored are going to be the largest components of keeping information safe. Creating safe places for data to be stored is our main priority. Access control is a great first step in mitigation of leaking customer’s data. We will start with the principle of least privilege. Only allow users to access what is absolutely necessary. Also, choosing to host all vital data on cloud based storage, instead of on a local server adds additional security.