

*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 | 02/20/22 | Kelly Illescas | Recommendations for The Gaming Room |

**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 game *Draw It or Lose It* is currently an Android-based game. The goal for CTS is to adapt the game for use as a web-based version. For this, a software application must be designed, and must include certain features. These features include the ability to have one or more teams, with each game and team having multiple players assigned to it. The team names must be unique, as only one instance of a team name can be used at any given time. Additionally, only one instance of a particular game can exist at a time. To do this, CTS will need to ensure that each instance of a game, team, or player has a unique identifier.

## [Design Constraints](#_2et92p0)

The design constraints include those for unique identifiers. More specifically, unique identifiers must exist for each player, each team, and each game so that only one instance exists in memory at any given time. The issue presented here is that code must be developed which allows the game to check existing players, teams, and games, and not allow an identical name to be implemented. In addition, the game currently only exists in an Android-based environment. The code will have to be rewritten to function in a web-based environment. This will require development using a new language, especially as the new environment is to be adapted to other platforms.

## [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.>

In the UML diagram below, you will see that an Entity class must be created. This is a parent class, and its three children, Game, Team, and Player classes, inherit from it. Encapsulation is seen here, as both id and name can be derived from Entity for all three of the child classes. Even though these classes inherit from the Entity class, they need to be separate classes because no one instance of any identical game, team, or player can exist simultaneously with another. The ProgramDriver class is used to drive the multi-class program. It uses the SingletonTester to ensure that only one instance of a game, team, or player can occur at the same time. The GameService class demonstrates the portability of the other classes, meaning that each class can be identified if it contains a bug or other issue. The “0…\*” connecting the GameService class to the Game, Team, and Players classes signify the association of zero or more objects in each subsequent class.

**"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** | + Tools for design,  development,  programming,  and testing all on  one unit  - Limited hardware  options  - Costly | + Reliable  + Flexible  + Cost-advantage  - Not as widely  used, therefore  less ease of use  - Less support | + Most widely-  used, therefore,  ease of use is  higher-rated  + Software more  readily available | + Commonly uses  REST APIs  - Costly  development  (since it is usually  developed for both  Android and iOS)  - Costly to maintain  - Coding required is  lengthy and time-  consuming |
| **Client Side** | + Generally  considered to be  more secure than  Windows  - Costly  - More expertise  required than  Windows | + Considered to  be the most  secure OS  + Less costly than  other OSs  - More expertise  required than  with Mac or  Windows | + Less expertise  required than  with Mac or  Linux  + Less costly than  Mac  - Generally  considered to be  the least secure  of the 3 OSs | + Updated  frequently  + Generally better  speed and  performance than  the 3 Oss  + Can sometimes be  available offline  - User required to  download app onto  device  (disadvantage  because  sometimes user  may not want to  user more storage) |
| **Development Tools** | * Swift * C * C++ * Objective-C * Xcode is the IDE of choice * Eclipse, VisualStudio are also used   Use of these development tools and IDEs require a team that is well-versed in different programming languages and editor resources. Additionally, most commonly used IDE, Xcode, is free to install, but requires a yearly fee of $99 to upload to the App Store. | * C * Python * C++ * Emacs, Vim, and VS Code (editor) most widely used * Other IDEs used (although more rarely) are PyCharm, Atom, and Eclipse   Use of these development tools and IDEs require a team that is well-versed in different programming languages and editor resources. Additionally, most commonly used IDEs, Emacs and Vim, are free. | * C++ * C# * Visual Basic * JavaScript * VisualStudio is the IDE of choice * Other IDEs used are NetBeans, Eclipse, PyCharm, and RAD Studio   Use of these development tools and IDEs require a team that is well-versed in different programming languages and editor resources. Additionally, most commonly used IDE, VisualStudio, costs between $45 and $250 per month to use | * Java * Kotlin (for Android) * Swift * Android Studio, Eclipse, VisualStudio, Python Suite – all widely used IDEs   Use of these development tools and IDEs require a team that is well-versed in different programming languages and editor resources. Additionally, most commonly used IDEs have the following costs:  Android Studio: $25  Eclipse: $0  VisualStudio: $45-$250  Python Suite: $0 |

## 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 believe that Windows OS would be the most appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.
2. **Operating Systems Architectures**: Windows is known to be the most versatile operating platform. It comes in two architectures: 32-bit and 64-bit. The 64-bit Windows is the preferred option for gaming, as it can support many windows running at once, as well as the fact that it contains more memory than the 32-bit. However, there are factors that need to be considered when choosing to utilize the 64-bit system, such as amount of RAM, Windows version, security, and other hardware considerations. The 32-bit Windows does not offer as many features and is slower to perform, but is more compatible with older hardware and software.
3. **Storage Management**: Storage Sense is a key feature for Windows storage management. It is ideal because it can do many things behind the scenes such as empty the recycle bin, clear the download folder, and delete temporary files. The user can either have all of this done on a schedule, or can select an option to perform the actions immediately. With Storage Sense, you can also target individual aspects of your hard drives, and clear things out that are taking up too much space and causing your machine to be sluggish. For instance, the Apps & Features section typically contains a lot of data, and many users find that there are apps that they are no longer using that, when cleared, could free up space on their hard drives.

For the user who does not feel comfortable deleting files or apps, CCleaner is a great option to

help him or her decide what to clear. The program will scan the system, and send the user a list

of options and explanations for freeing up space. Another plus side to this is that the program is

free.

1. **Memory Management**: To manage memory for *Draw It or Lose It!*, Windows uses virtual memory. Virtual memory is built into the system, but at times virtual memory can be low due to the constraints of gaming, especially graphics. To combat this, adding more RAM is beneficial, but can be costly to the user. An alternate option on the developer side could be to utilize a storage API. This way, we could employ cloud storage and manage the image file storage from our end.
2. **Distributed Systems and Networks**: Using a game engine is an ideal way to make our game usable on multiple platforms. Game engines support both 2D and 3D development. The first thing we developers must do is ensure that the game engine we use is cross-platform friendly. There are several options available nowadays, which wasn’t the case in the recent past. These engines are equipped to communicate data across various platforms while simultaneously preventing issues with connectivity and outages. Some examples of successful cross-platform engines we can consider for *Draw It or Lose It!* are Unreal, Unity, CryEngine, GameMaker Studio, and LibGDX.

There are pros and cons to each engine. We must explore these pros and cons and decide what is best for developing our game across our desired platforms. For instance, Unreal is beneficial because it uses a visual scripting tool to design the visual elements of the game; the drawback is that it also requires the developer to install separate SKDs for each desired platform. Unity offers 2D development, which is sufficient for our needs, and allows us to easily choose the version and platforms we desire. The required SDKs are automatically downloaded, and Unity is available for mobile development. LibGDX is a free option, and uses Java. Its API works across all our desired platforms (Windows, Linux, MacOS, and mobile). Its drawbacks are that it isn’t as powerful as Unreal or Unity, and that it doesn’t come with a GUI. GameMaker Studio is powerful, but requires a professional version to enable cross-platform development. CryEngine contains powerful visuals, but direct changes to the code must be made in order for it to work across platforms.

As far as preventing standard outages or connectivity issues, a strong server is required. This would aid in allowing large amounts of players at one time, while also providing a backup system to combat outages.

1. **Security**: Windows XP and beyond has its own built-in security system, Microsoft Defender. This is a wonderful, basic security feature, whose benefits are that it comes installed for free and contains a scanner for email, internet, and the cloud. However, it does not protect all of these systems; therefore, additionally security features must be enabled.

Microsoft SQL servers have been developed for some time, and are rapidly gaining the ability to enhance security across multiple platforms. There are many mainstream editions of the Microsoft SQL server, but more recently, specialized editions have been created to target specific platforms, such as gaming. For the gaming industry, Microsoft Azure is a wonderful SQL system. It is cloud-based, scalable, cost-friendly, and securely managed. There are other options, such as Amazon AWS and MySQL, but because we are interested in developing on Windows, the Microsoft option is ideal.

Another option to help combat security issues on the user end is to enable a single point of presence (SPOP). The SPOP allows the user to be logged into only one instance of the game at a time on his or her devices. In other words, if a user logs into the game on a mobile app, the previous login on his PC would be automatically logged out to he could sign in on his phone. Using the SPOP would have two benefits: keeping a user’s login secure, while also ensuring that game progress is saved correctly.