

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 4**](#_Toc115077323)

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/20/2023 | Kody Greenberg | Added summary, requirements, constraints, model. |
| 1.1 | 02/02/2023 | Kody Greenberg | Evaluation updated |
| 1.2 | 02/13/2023 | Kody Greenberg | Reference updates, added recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room (TGR) would like to expand their game “Draw It or Lose It” to more platforms than Android that it currently operates on. The game works by pulling up a blank canvas and it begins drawing a picture from a large stockpile of images. A game consists of four rounds lasting one minute each, multiple teams, and an image is fully drawn in 30 seconds. If a team doesn’t guess the image, the other opposing teams have 15 seconds to solve with one guess each team. Each game will have one or more teams, multiple players per team, names that are unique, and only one instance of the game can exist at any given time.

## Requirements

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Runs on multiple platforms.*
* *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)

The major issue on the technical side for TGR is they wish to use their working Android game to facilitate play across multiple platforms including other mobile devises as well as Mac, Windows and Linux systems.

We can meet these technical constraints by having a few dedicated teams to port over the working Android code to other mobile devices possibly using a language like Java. We can also port Java to a web-based coding scheme to make it usable on PC devices.

Some business constraints to think about are software licensing and legal considerations. Since it is based on the 1980’s tv show, is there any breach of copywrite? Is there any software that was developed for their Android app that has been copywritten?

The business constraints will be handled before development takes place to ensure our legal right to use and port over their currently working application. This should be handled by clear and concise meetings with the executives and legal departments.

With the business constraints settled, we need to think about the implications of the design constraints on our application development. This would mean we have a few teams working simultaneously to accomplish this work. We may have dedicated teams to work on the mobile port over and the web-based distribution environment. They should work alongside one another to ensure that the programs work efficiently and similarly. No major differences should be noted, the game should play and function as closely as possible to each platform. This would also mean we have servers set up to handle the different games the users create. A final thought is to let the users know that a minimum internet speed would be required - likely in the 3-5 Mbps range.

## [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 Entity class creates a relationship between the Game, Team, and Player Classes. So all will inherit or obtain some sort of information from Entity. In this UML we see this inheritance. The Game, Team, and Player classes will share information like “id” and “name” making the Entity a super class. Looking at Team and Player classes they are a relationship type called “has a.” While Game has a Team and GameService has Games. In a UML this is known as Aggregation (HAS A). This indicates that an instance of one class has a reference to an instance to another class. Looking at the UML again, we see GameService has a reference of Game, Game then has a reference of Team, Team then has a reference of Player. Each reference from game on, can call multiple copies of the classes so that we can make multiple teams and those teams have multiple players. GameService must be capable of making many games but recall only one can be stored in memory at a time. Which is where the singleton ProgramDriver and SingletonTester classes come into play.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Flexible terminal commands to configure the server, access or make changes securely.  **Advantages:**  Upgradable, Various choices for web hosting.  **Disadvantage:**  Less preferred for web hosting.  **According to Apple, Mac OS server has been removed as of 4/22/2022**(Apple, 2022) | Linux is built more secure and a preferred choice among most.  **Advantage:**  Security flaws are exposed before becoming an issue. More control of servers.  **Disadvantage:**  Difficult to find applications that support web hosting. | The most dominating platform currently.  **Advantage:**  Most software is available with low loading time.  **Disadvantage:**  More likely to be Virus prone, High cost for servers. | More popular and highly portable.  **Advantages:**  High portability. Cost-effective. Widest reach.  **Disadvantages:** Poor security on most. Highly selective of either Android or iOs. |
| **Client Side** | Moderate expertise and time. Cost similar to Windows but likely to be slightly more. | Most amount of expertise and time required. Least server cost among the choices, but highest maintenance costs. | Least amount of expertise, time, and training. Server cost is more than Linux but maintenance is less. | Most flexible to the client. Can see updates anywhere. May be difficult to implement compared to a web-service. |
| **Development Tools** | While Macs can run all the most common programming languages, Swift is the popular choice. Java may be usable to minimize work scale, but third-party software like VS, Eclipse or Code Runner are required IDE’s. | Virtually any language can be used on Linux. It can take advantage of Eclipse, VS, and other IDE’s. | Windows seems likely to be the easiest to run and includes support for the same coding languages and IDE’s as Linux. Popular options are VS, Eclipse, VSC, PyCharm, and many more. | Applications are typically made in the Swift language for iOS and C# or Java for Android. These languages can be migrated to any of the three OS discussed if needed. |

## 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 building “Draw It or Lose It!” on Windows OS. Even with the level of server costs it is the most dominant of the three PC options. With the relative low expertise and time, the cost will balance out and cause it to make more profit in the long run. There are plenty of useful IDE’s we can choose from to ensure the project is completed within a given reasonable schedule. It can also support cross-platform gaming with a proper game engine.
2. **Operating Systems Architectures**: Windows provides useful services utilized by all windows-based applications. It can also facilitate virtual environments and use cloud storage efficiently. It comes with a well-designed Graphical User Interface (GUI), while accessing system memory and storage efficiently. This also would refer to graphics, multimedia, messaging, and web services. These can be utilized with a user account or a specific server.
3. **Storage Management**: One option would be to utilize windows’ ability to add more storage to a system if we have the room and need by adding extra Hard-Disk Drives (HDD) or Solid-State Drives (SSD). Then we could merge it into a storage pool. This will improve performance, protect data, and allow for additional storage spaces should we need to expand even more in the future.

My recommendation for storage would be to utilize Cloud storage and only pay for exactly what we need. This would reduce waste and be overall less costly. This would off put the need for maintenance, space for our own servers, and increase the ability to expand without notice onto the chosen cloud storage company. Paying for a cloud-based storage system would include adding their security measures to the storage and prevent downtime from the servers going offline.

1. **Memory Management**: If we hard code everything, we could cause the system to operate slower. Instead, we should choose a game engine that may offer a large library of pre-built code to help us efficiently build a system that works quicker and with less strain on the OS. We could keep a database of our library files within our servers which would be used to help organize these files. Using a game engine would also allow us to employ virtual memory so there is even less strain on the device OS. We would also want to only pull the image we need for that game and not pre-load all 200 images. Using a Content Delivery Network (CDN) such as KeyCDN would be a good way to save system memory. (proinity, 2023)
2. **Distributed Systems and Networks**: While we would be choosing one OS to program in, the cross-platform game development tool can be the very clear answer to assist in distributing this game to as many people as possible. One widely used and cost-effective cross-platform game engine is called “Unity” which can support Windows, Linux, Android, iOS and more. (Technologies, 2023) Finding the right server to use is key. Along with redundant backups from cloud storage, we can minimize any downtime to mostly just our patches or updates. We should plan to have as many people play as possible. Of course, this is not an easy task since there may be thousands of games with several users in each game. So, we need to plan for a strong system with good connectivity. That is why I recommend we employ as many cloud-based services as possible. Doing so will keep a very high uptime and minimize any issues we could create. Deciding on a server that supports many users, cloud-based services, and the ability to handle high traffic will lead to a very healthy game environment where users will be satisfied. With these in place, if the user has internet, we can have nearly 100% uptime and playability.
3. **Security**: Windows has its own software regarding security. This built-in program may perform well, but there is other software available. Malware and virus protection software may be used to protect servers and company assets like computers. The programming of the game should also employ the best in standard practice of security. Using role-based access along with administration rights will need to be implemented. Encryption of data from users like personal names and information will also be key to implement. In this way, even if a breach happens our users will not have to worry about their personal information being taken. While windows OS is the widest used of the options, and likely to be hacked more due to that, it has advanced security features and can be protected with proper preparations.

**References:**

Apple Company, (2022, April 22). *About macos server 5.7.1 and later*. Apple Support. Retrieved February 13, 2023, from https://support.apple.com/en-us/HT208312#:~:text=macOS%20Server%205.7.-,1%20and%20later,the%20app%20with%20macOS%20Monterey.

Technologies, U. (2023). *Unity Games Solutions – create 2D and 3D games.* Unity. Retrieved February 13, 2023, from https://unity.com/solutions/game

LLC, proinity. (2023). *Cdn Storage*. KeyCDN. Retrieved February 14, 2023, from https://www.keycdn.com/cdn-storage