

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**

**Domain Model 4**

**Evaluation 5**

**Recommendations 8**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/18/2023 | Justin Hancock | Original Release |
| 1.1 | 02/02/2023 | Justin Hancock | Evaluation added |
| 1.2 | 02/13/2023 | Justin Hancock | Recommendation added |

**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 is searching for a group to facilitate the development of a game called Draw It or Lose It. The game is played with teams made up of multiple players. The game will have a large library of drawings that will be revealed to the player in a timed manner while they attempt to guess the phrase, title, or thing the drawing represents. The game lasts four rounds and if the team does not guess correctly in the first 30 seconds; then, the next team has an opportunity to guess. They would like us to develop the environment for a web-based version of this game. Since the game is currently available on Android, we will be able to utilize that version to convert into the web-based version. With the requirements from the client, I believe the Singleton Design Pattern will be the best option for the creation of this game.

## Requirements

Software 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)

The first design constraint is cross platform compatibility. With the application currently being only on Android, we need to make sure that the requirements of the gameplay are consistent on both operating systems.

The next design constraint is the gameplay itself having one or more teams and multiple players per team. We must make sure that the servers can handle running multiple teams and players.

Unique game and team names are the next design constraint. To resolve this constraint, we would need to account for the teams and players having only one unique name. Only one instance of the game can exist in memory at a given time. We will need to strictly manage memory allocation for these instances. With the game being cross platform, compatibility between usernames on both will need to be considered as well for ease to the user.

Lastly, the game will have a large storage filled with drawings. We can adopt the drawings used on the Android application, but we will need to consider any potential legal issues with use of these drawings on this platform along with any newly added drawings.

## [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 UML Diagram below gives you a visual representation of the game itself. The first item that I will mention is the ProgramDriver and SingletonTester classes. The visual representation just shows that the driver class will use the singleton method to test the program. The singleton method is used to ensure that we only have one instance of the game in memory.

The next topics I would like to discuss is the act of inheritance, encapsulation, abstraction, and polymorphism. These 4 components make up the object-oriented principles that will make this program run fluently. Inheritance is used from the Entity class to the GameService, Game, Team, and Player class. Inheritance simply means that the attributes in the Entity class can be used by the GameService, Game, Team, and Player classes as well. Encapsulation is the representation of private and public attributes. The attributes listed below that have a “– “sign are set as private meaning that only that class can use that attribute, it does not pass down to the other classes. The attributes with a “+” sign are public attributes and can be used by other classes. Abstraction is shown in this diagram by only revealing the information that need to be shown. It gives the developer the option to add more detail when needed, but for now to show a complete picture without any confusion. The idea that different types of objects can flow through the same interface is known as polymorphism. This is represented in the class GameService by having many attributes that cannot be accessed by the parent or any other class.

The GameService class hosts most of the complex methods and functionality of the game itself. The other classes represent the unique game, teams, and players that are required for proper game play. The numbers on the lines between the boxes represent the number of associations between the classes. For example, Game can have 0 to infinity Teams, and Teams can have 0 to infinity Players.

**"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)

Below are characteristics, advantages, and weaknesses of each operating platform. I will be keeping in mind your requirements throughout this evaluation. Since the program has already been written for Android in Java, we could potentially translate that code to working on multiple platforms with minimal changes. We could use the same Integrated Development Environment for each platform which would save money. I believe we could also save some money by having the same team work on this project instead of multiple teams. Timeline would be the largest deciding factor on this decision.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | As of April of last year, Apple has discontinued their Mac OS X server; however, they are still available for purchase. The biggest strength of using Mac is the graphics. The weakness of using Mac is it can be very expensive, and typically doesn’t work well for larger companies.  The cost to purchase one of these servers is around $350.00. Since being discontinued, the OS X Server can come as an add on to any Mac and it only costs about $30.00. This cost can grow based on license amounts. | Linux is known for their free and open-source software. Companies like Google, Amazon, and Facebook are switching to Linux servers. The security and Stability of these servers is well known. The hardware does not need as many updates.  Overall, Linux has a lot of the tools that you would need to create a program here; however, it does require expertise to work with this operating system. | Windows servers can store an incredibly large amount of data and can max out to 24TB of RAM. They use a Graphical User Interface which is the buttons and icons you can select.  Licenses and cost of the actual hardware for Windows can be very expensive depending on the size server and how many clients the server is hosting. | Mobile devices now are basically a remote computer. You can use it to search the web or save files, but you cannot use a mobile device with multiple users. They do not have as much storage or RAM as your typical server.  A mobile device would not work for a server. A server needs to be hardwired to not lose service and lose the whole system. |
| **Client Side** | The same web browsers are offered on Mac as other operating systems. To reduce costs, time, and expertise we could use a similar language for each platform. We will need to make sure that the client is running the latest version of software. From a market perspective, Mac is on the lower end of overall use which will reduce the opportunities with this build. | Since we will be attempting to write a program that is similar across all platforms, the code written for Linux could easily be written in a common language like Java or Python. Linux is not typically a widely used platform so the development may not return the revenue you would desire. | Windows is the most widely used operating system for users which means that development on this OS would likely produce the most revenue. C# is the most common Windows development programming language. | Java is the main application development language for Android. Swift is the main application development language for iOS. For Android, Windows and Linux would be the best bet on hosting the application. iOS would have the same requirements that running the program on Mac would have. |
| **Development Tools** | You can use just about all programming languages on all operating systems. Visual studio code is an IDE that stands out to me because I have used it as a beginner, and I know it is available on Mac, Windows, and Linux. VS Code costs either $45 or $250. If you are writing the program in similar coding languages, you can probably work with one team condensing the cost. | If we chose to write the code like Mac, as stated before, VS Code is available for Linux and could be used and no additional cost be used for this server. | Visual Studio has an optional ad on to the IDE that will add the C# programming language into the IDE. We are still consistent on the IDE available to use for this project saving us in overall cost. The programming language itself is different so we may need another team depending on how quickly the program needs to be completed. | Java and Swift can be written in Visual Studio. Java is a normal programming language for VS, but Swift is an ad on language. We may be looking at the more expensive side of the IDE, but it would allow us to write all the code across all the programming languages in the same IDE. |

## Recommendations

From the beginning, the goal of this game is to create an environment for the development of a web-based version of Draw It or Lose It. The following topics list my recommendations for the gaming environment that you should create based on potential revenue, overall costs, proper structure, and security.

1. **Operating Platform**: Since our main focus is to create a web-based version of the game, I would recommend using Windows Operating System. With Windows being the most popular operating system, the potential revenue just by choosing Windows multiplies exponentially. We will have some up-front cost to get the proper hardware and possibly some licensing costs as well, but Windows will also open us up to expand from just web-based to other platforms as well when the time comes.
2. **Operating Systems Architectures**: Windows Operating System is easier to use with the built in graphical user interface. The Integrated Development Environment, Visual Studio, is available on Windows and one of the most popular IDE’s on Windows. We can use that program to write the code in multiple languages when you decide to expand from just the web-based version of this game.
3. **Storage Management**: There are three options when it comes to storage for this game. You could have a server with an Hard Disk Drive which now is a dated option. It would still run the game perfectly fine, but depending on the amount of users the game could run slower because of the way the HDD runs the memory. A Solid State Drive would be the option that I would recommend. It is faster than the HDD and significantly less expensive than the third option, Windows cloud server. My recommendation would be to build the game on the SSD to begin with and let some of the revenue from the game development lead you into a purchase of a Windows cloud server to expand to other platforms.
4. **Memory Management**: Random Access Memory is the physical element that uses memory on a computer. Virtual memory is like a storage room that holds files to get when the computer runs out of RAM. I recommend using physical memory. The largest option for RAM in a server is currently 16 TB. To put this into perspective, most regular home computers or laptops have between 8 and 16 GB of RAM. I’m not sure that 16 TB would be the appropriate amount of memory; we should be able to use less, but memory is very easy to change in and out. We could begin with a smaller chip for the RAM and expand to a larger one when needed.
5. **Distributed Systems and Networks**: With Windows Operating Systems, there are a few things you should know about communication between other devices and the dependencies within the systems and networks. Of course, you can always have power outages that could cause a server to shut down. To prevent these from causing any types of permanent damage, we can have battery back-ups along with generators to keep things running in case of power outages. Since the environment will be shared amongst the players, as developers, we will need to make sure that the communications between the players is flawless. This will alleviate potential lag and ques for the player so they can play with no wait times or pausing issues while playing.
6. **Security**: There are always threats to your information. To prevent unauthorized access to your information in this game. We will make sure that we have an appropriate antivirus software set up to warn us against attempted breaches. It is important to have a daily routine to check on potential breaches. As developers we will write the program to have built in encryptions on the user information so it cannot be accessed by anyone without authorization. If we make sure to update our software as often as possible and stay on schedule with maintenance, we will be doing our best to make sure everyone’s information is as secure as possible.

One piece of security that is always difficult in video games is cheaters. There will always be someone who wants to win bad enough to cheat. We can run anti-cheat software with the game that will, most of the time, find a cheater and allow us to remove that person from the game.