

Draw It or Lose It

# **CS 230 Project Software Design**

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 6**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/13/2023 | Aaron McDonald | Updated the Evaluation Section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room currently has an Android App called Draw It or Lose It that is a game similar to that of Win, Lose or Draw. The goal is to expand this game to be a web-based game as well that will have one or more teams involved in the game. Teams will be comprised of multiple players, each team and player having a unique name to avoid any overlapping. Lastly, only one instance of the game can exist at any time to prevent any overlapping of data. All of can be accomplished by creating a web server that will host the game and serve clients. This will require a server platform to work on such as Linux or Windows and requires using what The Gaming Room has already created for the game or the games API. Lastly, a singleton design pattern along with unique ID’s for team and players will be used to ensure that no instance of player, team or game are the same.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Having the game be in a web based environment does imply some amount of constraints on the game out of the box. The first major constraint will be to decide which operating system the server will be hosted on and which operating system the expected clients will be using to interact with the game. This will determine how the environment needs to be developed but also determines how accessible our game is to users of different operating systems. Another constraint will be the expected access to internet connection. If we expect our users to have an average internet connection, then we may need to consider what resources we use in our game. For example, if the internet connection is expected to be slower, then we can’t have a lot of large assets such as complex 3D models or videos. There may also be drawbacks to not having a graphically advanced game since the expectations from modern users would be to have better visuals, especially in games. This may impact design choices as we need to decide what assets are bare minimum and which add to the overall experience.

## [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 shows how the Environment relative to holding the different games will look to provide the expected functionality. Firstly, the Entity Object acts as the parent object for the Team, Player and Game objects to inherit from. Inheritance is an Object Oriented Principal that allows Objects to all refine the implementation of a parent object while all still maintaining the underlying functionality. In this case the three classes mentioned all carry the ID and Name as well as the accessor methods for these attributes. This is helpful as each of the objects needs to have a unique ID and a name associated with them. Next, that GameService Object acts as our singleton design pattern. This object works by ensuring there is only ever one instance of itself in memory. Assuming that all of the games, teams and players have unique ID’s associated with them, this ensures that there are no duplicates of any of these objects in memory as well. GameService then handles all of the creation and fetching of games, players and teams to ensure that only one of each exists. Lastly, the ProgramDriver and SingletonTester each act to support the overall program by actually driving the program.

**"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** | Advantages:   * Security * Reliability * GUI * Ease for Mac Clients * LDAP & ADP   Disadvantages:   * Cost for hardware * Software Licensing | Advantages:   * Cost / Free – Open Source * Reliable – very few updates * Security * LDAP & ADP   Disadvantages:   * Software compatibility * Requires more expertise | Advantages:   * GUI * Software compatibility * LDAP & ADP   Disadvantages:   * More Security threats * Software Licensing | Advantages:   * Mobile * Free Software   Disadvantages:   * Security / Social Engineering * Low Resources * Low capabilities * Not very well supported |
| **Client Side** | * Low user volume, not many gamers * Proprietary hardware is required for use * Low expertise to use the software | * Low user volume, not many gamers * High expertise to use the software * Increased time sink / software compatibility | * Higher user volume, most gamers * No proprietary hardware * Low Expertise to use the software | * Higher user volume, everyone has a cell phone * Increased time and cost for touch screen capabilities * Low expertise to use the software |
| **Development Tools** | * Oracle for Java Applications / Maven * Pycharm for any Python * DbVisualizer for Database work | * Eclipse for Java and C++ * Visual Studio for most programming languages / free * Code::Blocks | * Oracle for Java Applications / Maven * Pycharm for any Python * DbVisualizer for Database work | * Visual Studio for Android Apps * CppDroid for coding with C++ * Xcode for IOS apps |

## Recommendations

1. **Operating Platform**: My recommended operating system for Draw It or Lose It would be to use Linux as the Server host and then start by implementing the client side on Windows Operating system’s first and then expanding as needed. Additionally, creating this as a web-based application will make it much more accessible to all players except for those that use consoles.
2. **Operating Systems Architectures**: Linux is an excellent operating system to host the server from. Having higher security than the other options and being free makes Linux an incredible option that will be robust with very few system updates. Additionally, Linux can be installed on a machine without any proprietary hardware, meaning that the machine for the server can be picked to suit the exact needs of the software and no more than that. Additionally, this means that cheaper options may be usable if money is a major concern in some areas. Focusing on the web based side of this application and on the user experience for Windows clients specifically will allow for customers of our target demographic to have easier access to this app. Anyone who games on their desktop computer almost always uses a Windows Operating System since it is the best for gaming, therefore, ensuring that the client experience on Windows machines is the best will improve the revenue from the app. Creating this as a web-based app will also allow users of other operating systems to use the app because the web browsers will act as a middle-man between our software and the app experience, meaning that we would only need to develop the software for one platform, the internet, and then modify the experience slightly to cater for each browser or operating system used.
3. **Storage Management**: My recommended Storage systems would be to have two 512 Gigabyte hard disk drives at least. My recommendation would be to have twice the number of drives as the storage that you want as I believe we should run the storage in a RAID 1 array setup. This will ensure that if any drive fails, the information will be kept safe due to the redundancy of how the storage works in a RAID array. With the storage size I have laid out, that should be more than enough to store all of the image files from Draw It or Lose It. However, these also leave room for expanding the games further as well as storing user data. Lastly, on the software side of memory management, I recommend that the images be stored in a contiguous management system as this will allow for the fastest fetch times for the randomly selected images from storage. Since these files shouldn’t need to be over written or removed from storage, only ever read, this should not be a problem to allocate the required memory.
4. **Memory Management**: As for the Memory, I recommend at least 32 Gigabytes of fast RAM or even more if possible. RAM will act as the repository of information for the programs during runtime. Having, thousands of clients simultaneously means that we will need a lot of Memory to be able to manage all of the images that will be copied from storage and used in the game. On the software side of memory management, it will be important to check for any memory leaks as these could rapidly increase memory usage to a point of usability. It is of the highest priority that all temporary memory gets deallocated as soon as it is no longer needed.
5. **Distributed Systems and Networks**: As I have already specified, having a web-based application will allow any number of different operating systems to use our application. This is because we would program an HTML, CSS, and JavaScript web-facing front end, while most of our functionality can remain on the server we build. Furthermore, we could purchase online server space and time as well, such as AWS which would save resources on us needing to maintain our own servers. The major drawback of this approach to creating a backend is that we are no longer in control of the maintenance, nor do we have direct access to these servers. This means that if there is a problem with our server, we have to wait for our third party distribution system provider to fix it, potentially causing a loss of business. At the very least I do recommend creating a web-based application as this would allow users of most operating systems to access our application. The only exclusion to this would be for console gamers since these do not typically have web browsers.
6. **Security**: Security is always of the highest priority and this is one of the reasons I recommend Linux as the host for our server. Additionally, on the software side, it is important to note that we should always have our software working with the minimum-security access required to accomplish its task. This is because if our software ever fails, it will do so in the least destructive way possible. Additionally if our software is corrupted by mal intent, it will have the least access to do any damage. As for hardware security, if a server is intended to be built, then having a two step authentication would be best for any access to the server, and again people should have the least security access possible to get their jobs done in relation to the server. The authentication could be a mixture of a password and a retinal scanner or perhaps fingerprints scanners. The two-step authentication could also be very useful for our clients to maintain security over their data. Having a single-use password generated and sent to the client for each login attempt, along with their normal password, would heighten security to a point of almost unbeatability.