

# **Draw It or Lose It**

# **CS 230 Project Software Design Template**

Version 3.0

## 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 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/18/2025 | Avrye Tompkins | Updating executive summary, requirements, design constraints, system architecture view, domain model, evaluation, and recommendations. |
| 3.0 | 8/16/2025 | Avrye Tompkins | Updating recommendations page. |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is based on a television game show, Win, Lose or Draw, circa the 1980s. The idea of the game is for teams to create a drawing, then have to guess the drawing. The application moves that idea into a game application. This application will render images for a large library of stock drawings and clues. There will be four one-minute rounds, with drawings slowly being rendered in up until the thirty-second mark, at which they will finish being rendered. If the original team does not correctly guess the prompt, the other team will have a chance to guess the prompt within 15 seconds. The Gaming Room does not know how to set up the environment. They also plan to create a web-based version of the app.

## Requirements

* The game will have at least one team with the ability to have more.
* Each team will have multiple players assigned
* Game and team names must be unique, meaning if a name is taken, then another user can not select it. The system will need to check each name.
* There can only be one instance of the game running at one time. This can be completed by creating unique identifiers for the game, team, or player.

## [Design Constraints](#_2et92p0)

Some constraints could be having either too few or too many players. Not having enough images so the game does not feel boring. Some technical restraints could be not having enough memory or space for the number of people and photos. Another is properly loading in different photos each time, without repeats in the same rounds. Unsure of the environment. Moving the application to web web-based version. Checking to make sure that a unique identifier cannot be repeated.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: Although nothing is required for these projects, this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be necessary 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 first is that the whole “The Gaming Room” is in the folder of com. gamingroom, so all classes are within this folder. The first thing to note is that the SingletonTester has a usage relationship with the ProgramDriver. This is noted by a black arrow with the <<uses>> in the middle of the arrow. We can also note that the ProgramDriver, which runs the whole program, also publicly contains the “main.” The SingletonTester class contains the public testSingleton. These two have a disconnect from the rest of the classes.

The next note is that four classes have a generalization arrow. The first class to look at is the entity class, which has a private ID and name. It also contains a private entity class, with there also being public entity IDs and names, a setter and getter for both names and IDs, and a toString. This allows for there to be only one instance of a name and ID to keep all information separate with no overlap. The Entity class inherits the Game class, Team Class, and Player class. The player class has a public ID and name, public toString. Players have a zero to infinite association with the Team class. This means that the class has infinite information for the players, zero information. The Team class has a list of private players. Next, the team class has a public Team ID and name, adders for the Player class, and a toString. The Team has an association with the Game class, with a zero to infinite association. The game class has a private team’s class list. It also has a public Game ID and name, adder for team names, and a public to string. Finally, the game class is only connected to the GameService class. The game service class contains most of the information, including name, ID, getters, and setters. For the attributes that contain all private games, nextGameId, nextPlayerID, nextTeamId, and service. These attributes contribute to keeping only one specific player ID per team, one name per person, and one ID per game, so there is no overlap. Next in the operations, we can see a private GameService class. Next are all public operations. This is Getinstance, addGame, getGame, Getgame, GetgameCount, getNextPlayer, GetNextTeamId. This sets a new player ID and gets the next round of IDs for multiple games. All this information, once again, is encapsulated in the com. gamingroom folder.

**"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 paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Has high performance and is known for reliability. It has a higher level of security.  Backup features, i.e., Time Machine  More expensive than windows. X servers are $599 as they are right now. Requires the hardware as well | Cost-effective and open source. Often known for its stability and lack of crashes. It's also known for the scalability and growth of accommodation. With a high security level and great customization, and high performance.  Also known for its web hosting | Highly compatible with all types of apps and a larger range of software. It has decent security comparatively. Wide compatibility when it comes to hosting our application. Good scalability  Not as secure compared to other systems, such as Linux.  No integration with cloud hybrid environments. | The hardware for mobile devices often has heavy limitations due to size, trying to be the most compact.  While it is possible to host a server on a phone, it’s not the best option in comparison to computer-based servers.  Phones are not scalable due to the hardware, and the cost of hosting a server on a mobile device is not really known since it’s not typically done. |
| **Client Side** | Streamlined interface and user experience.  High price compared to computers  Smaller packages range from $29-$39 a month for smaller servers, and $45 to $55 for the larger server.  Harder to learn when not accustomed to the Mac suite.  Not ideal for gaming. | Linux can use many different programming languages. The cost is low, which makes it a budget-friendly option. Many learn the basics of Java, Python, or C++.  Along with these pros, Linux supports multiple players and the GNU/Linux servers. This makes it most in line with the Gaming room. | Users can find the interface complicated, but once learned, it can be easily used.  Integration with other Microsoft applications. | The mobile device hardware is especially streamlined and easy to use, with most people already having access to one.  A direct cause of this can be seen as many creators make applications meant for IOS or Android.  Where it’s easier for smaller projects they are not designed for multiple users. The limitation is the amount of RAM a mobile device can hold, making it hard to expand. |
| **Development Tools** | Requires hardware for the X server  Supports Apache, PHP, NGINX, NeXTSTEP, UNIX, java, cocoa, carbon runtime.  Also offers Appium, which can check to see how hybrid games will work on phones and other devices, but also at a cost. | Relevant programming languages include Apache, Nginx, and LiteSpeed. Also contains content management systems like WordPress, Drupal, and Joomla. Along with these programs, there are also the basics of JAVA or C++, along with Python. | Some relevant programming languages are ASP.NET and Microsoft SQL.  Windows servers often have a large variety of add-ons that allow the creation of hybrid apps. This casts a larger net to allow the game to be downloaded and used. | Most mobile devices use hybrid apps for cross-platform gaming to keep up with the other competitors. It also has to be considered that most people use a computer, so implementing a cross-platform language is recommended.  Some languages used are Objective-C, Swift, Kotlin for Android, and Java. You also have to consider which type of device you want to implement and find a language that works well and can be implemented into as many as possible. |

## Recommendations

1. **Operating Platform**: I would recommend the operating platform Linux. It is one of the highest-ranking systems within the graph.
2. **Operating Systems Architectures.** Not only is it a reliable and cheap option, but it also has scalability. Linux also has good security and is in league with its competitors, such as Windows. It can also use multiple different coding languages, which means it can adapt to multiple different platforms. Linux also supports multiple players with the GNU/Linux platforms.
3. **Storage Management**: For storage, I would recommend a type of cloud storage. This will allow another team from outside of The Gaming Room. While it could be a little costly, the money is made up for by using a cheaper operating system. I would recommend Nextcloud, which is compatible with Linux. It is a “self-hosted open source cloud file storage technology that is a perfect fit for the privacy and security” of win it or lose it. This is also aligned with the game room because it can scale with new gamers. Storage itself works by saving the photos in the cloud, which can then be accessed. This storage management system allows for enough space for all photos to be saved, plus enough space for the game and any possible growth.
4. **Memory Management**: The operating platform uses memory management is by different types of memory allocation. There are two types of allocation, static and dynamic. Stataic compiles the assets preloaded into the game, such as username, games, and any other constant information. Dynamic loads information as the game goes on, which includes the photos, questions, and scores. These help save on memory while the game is running.
5. **Distributed Systems and Networks**: The way this is accomplished through the software is by having multiple types of programming languages. This allows for the best type of language to be used, while also being flexible with different platforms. There are also different types of network connections by using a cloud-based storage management system. Cloud storage can range from multiple different countries, allowing for there to be constant connections as long as there is internet. Since the game can be put online, along with cloud-based storage, the game can run from any platform.
6. **Security**: Regarding security, when it comes to storage, we can see that Nextcloud ranks with security as its main focus. There are also many features, including multi-factor authentication and protection from brute force hacking. As for the game itself, the user's information can be behind an authentication screen that requires a specific username and password to access. On top of that, the more confidential information can be temporarily stored in the game, then soon deleted after it's been used. This keeps the information safe after the user logs off a session. The security on both storage management and Linux ranks high, with the capabilities constantly being tested.

**Citations**

**Bonheur, K. (2025, March 31). *Advantages and Disadvantages of MacOS*. Profolus.**

**https://www.profolus.com/topics/advantages-and-disadvantages-of-macos/#google\_vignette**

***HostMyApple: Mac VPS cloud hosting of MacOS Sequoia Servers*. (n.d.).** [**https://www.hostmyapple.com/mac-vps-hosting**](https://www.hostmyapple.com/mac-vps-hosting)

***🏆 How Nextcloud stacks up*. (n.d.). Nextcloud. https://nextcloud.com/compare/**

***Installation on Linux — Nextcloud latest Administration Manual, latest documentation*. (n.d.). Docs.nextcloud.com. https://docs.nextcloud.com/server/latest/admin\_manual/installation/source\_installation.html**

***MacInCloud - managed*. (n.d.). https://www.macincloud.com/pages/managed.html?msclkid=f8578a4616db132d4460e2c8712c693c&utm\_source=bing&utm\_medium=cpc&utm\_campaign=MIC%20-%20PAYG%20Managed%20Plans&utm\_term=macos%20server&utm\_content=MIC%20-%20Ventura%20-%20Canada%20%2B%20US**

**Muchmore, M. (2024, September 16). macOS vs. Windows: Which OS Is Best? *PCMAG*. https://www.pcmag.com/news/macos-vs-windows-which-os-really-is-the-best**

**Team, E. (2025, June 7). *10 Best Mac Developer Tools for Engineers in 2025*. ClickUp. https://clickup.com/blog/mac-developer-tools/**

**Teknoding. (2024, November 16). 15+ Pros and Cons of macOS/Macbook (The Ultimate Guide).**

***Teknoding EN*. https://www.teknoding.com/2024/11/pros-cons-of-macosmacbook.html**

**Whatley, S. (2024, March 27). *Comparing Mac and Windows servers for web hosting*. MamboServer.** [**https://www.mamboserver.com/blog/comparing-mac-and-windows-servers-for-web-hosting/**](https://www.mamboserver.com/blog/comparing-mac-and-windows-servers-for-web-hosting/)

***Windows Server*. (n.d.). http://peerspot.com/products/windows-server-pros-and-cons**