

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | 10/04/25 | Ridge Cox | This is version two of the development plan for *Draw It or Lose It.* The goal of this version is to revise and improve the development plan discussed in version one. We will discuss the server and client sides of the application; additionally, there will be guidelines listed for what development tools would be best suited for *Draw It or Lose It.* |

**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 expanding *Draw It or Lose It.* Pursuing this plan will allow The Gaming Room to enlarge its player base. The expansion includes implementing Web, Mobile, Linux, Chrome, and Mac versions of *Draw It or Lose It.* It will be important to keep the game homogenous across multiple platforms so users can easily navigate the game without confusion. Currently, *Draw It or Lose It* only works on Android; this means revisions will need to be made to the app so it can function on any platform. This will require modifying the original development of *Draw It or Lose It* so it is versatile with any user and compatible with any operating system.

This can be achieved by:

* Focusing on compatibility
* Ensuring the design of the game works similarly across all platforms
* Successfully choosing the most optimal IDE (Integrated Development Environment)
* Deliberating on the correct server-based development method
* Reviewing advantages and weaknesses of each platform
* Deciding on multiple development teams or one large force constructing the project
* Staying on budget

To ensure all the above, we will need to understand how servers impact multiple operating systems. Different operating systems will be more compatible with servers than others. This requires research on which operating system is the best fit for all current and future applications of *Draw It or Lose It.* Additionally, there are licensing costs for each operating system. This will be a crucial factor in determining the best operating system that fits within The Gaming Room’s budget.

## Requirements

There are several requirements to keep in mind when developing *Draw It or Lose It.*

**Web-Based Architecture:** Must be hostable on the web through browsers or mobile clients.

**Scalability:** Must support thousands of players across multiple games without crashing.

**Cross-Platform Hosting:** Must allow users from opposing operating systems to play together.

**Database Management:** Needs to store data from user profiles, team data, scoring, and more.

**Real-Time Processing:** Must be able to render, track time, and receive user input in real-time.

**API Layer:** Use RESTful APIs to interact with servers via HTTP requests for data, scores, and teams.

**Security:** Should protect user data via encryption and additional authentications.

**Cost Efficiency:** Must follow the budget to avoid derailing the project.

## [Design Constraints](#_2et92p0)

There are several major design constraints that will need to be evaluated to perform across multiple operating systems. These changes would need to be within budget; this is a constraint in itself. One major constraint is scalability. Currently, *Draw It or Lose It* does not have the infrastructure to support three extra operating systems, as it is scaled to Android only. The server capacity would need to be greatly increased to allow additional operating systems, as the player base would likely expand greatly

Another constraint on design would be performance; there would need to be improvements to communication so that there isn’t significant latency when playing. This is due to the fact that drawing and guessing would be happening in timed environments. Users who lose connection or lag could cause their team to suffer losses due to lack of connection.

The current state of security cannot handle verifications for this many new users. There would need to be optimal improvements to authentication so that the system can keep up with verifying data and encrypting it. One potential problem could be crashing of the website due to the influx of accounts being created. There would need to be enough server space for multiple new accounts with validation requests so the game can have a healthy release to the public.

Finally, there should be improvements to optimization so that mobile devices can handle the complexities of drawing and guessing in real time. Mobile is particularly vulnerable due to a lack of processing power, so the game would need to work within those limitations.

A few suggestions to improve the state of these constraints would be to increase server capacity to allow for more user data to be stored. Another suggestion would be to use JSON to compress data so it is evaluated more quickly. An additional improvement would be using caching so that multiple requests aren’t made for frequently used data. Caching is when you store frequently used data in temporary storage locations so it can be accessed more quickly.

## [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)

**Entity**: Holds commonalities, including Game, Team, and Player objects. It acts as the parent class, and these classes extend Entity. Encapsulation is found due to the private fields, and abstraction only exposes what’s necessary so there aren’t walls of code.

**Game**: Represents a single instance of *Draw It or Lose It* and keeps track of participating teams. It allows for adding teams and provides the teams to other classes in the program. It contains inheritance, encapsulation, and abstraction, just like above.

**Team**: Represents an individual team, allows new players to join a team, and lets other programs access player lists for teams. It extends the Entity class, meaning it has unique IDs and names. Once again, it contains inheritance, encapsulation, and abstraction, same as above.

**Player**: Represents individual players within the game, holds player info like the unique ID and name, and allows the player to be added to teams. Inheritance, encapsulation, and abstraction are also found here.

**GameService**: Manages games, teams, and players. It stores all game objects, finds already existing games in the service, and assigns the IDs for teams, players, and games. It ensures only one instance exists using the Singleton pattern. It contains abstraction (it doesn’t show every class how the IDs are assigned, just provides them) and encapsulation due to private IDs.

**ProgramDriver**: Entry point of the program where the system is tested and demonstrated. It creates the GameService instance, creates players, and adds players to teams.

**SingletonTester**: Tests to make sure the Singleton pattern is functioning properly. It confirms only one instance of the game service is running, then prints the results of the Singleton test.

**"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** | Mac is Reliable for local hosting, great for testing. Additionally Mac has a large user base.  Some of its flaws are that it is limited with its ability to host large developments, additionally licenses are very expensive. Many of the features Mac has windows also has, however, windows has more overall | Linux is the most budget friendly and optimal of all platforms, it has great support for python, java, etc. Additionally Linux is known for running very quickly, meaning less downtime.  There is a large flaw, Linux requires much more experience than other platforms, its steep learning curve can slow down the development process if the developers do not have previous experience | Window has Microsoft IIS (Internet Information Services) which helps with hosting web-based applications, additionally windows has built in user authentication and management, finally windows has a great support system.  There are a few flaws to windows such as  High licensing cost. On top of that windows suffers from being very CPU intensive due to their infrastructure. It’s also possible that Windows’s Frequent update may get in the way of development, an additionally flaw is that windows is easily targeted by malware attacks. | Mobile devices have access to cloud servers allowing for easy access, mobile devices are designed to be low bandwidth, one great benefit is that phones can be used on the go. However, phones do not act as servers so there would need to be a third party for hosting.  A few flaws I found are that mobile phones must have batters charge to work; they are also very vulnerable to data leaks. They are also not scalable as they are meant to run single users, not whole companies. |
| **Client Side** | Mac Supports both safari and chrome for web usage, it also has a great support system, additionally many users are very familiar with mac computers as well.  However, a major flaw is that safari would require development on Mac hardware, which can be very expensive over time. | Linux works great on Firefox, chrome, and other browsers. It’s low cost for clients, and tends to have great performance, the efficiency is a big reason many love Linux  A flaw is that some graphics cards are not compatible with Linux, particularly for drivers. | Windows is the most widely used operating system; it also has great driver and hardware support. Windows has access to Microsoft services like teams, OneDrive, etc.  Some flaws include,  Security concerns due to Windows being a large target of viruses, additionally Microsoft can make it difficult to integrate. | Some pros of mobile devices are that they have easy access to the web while on the go and mobile devices have built in sensors for easy client use. Mobile has the largest global base, additionally mobile has strong access to the web  However some flaws include, mobile devices have  Small screens and limited storage, additionally they are not very flexible for UI use. They are not made to host large projects as they are compact and aren’t meant to store and run large files |
| **Development Tools** | Mac supports multiple development tools such as IntelliJ, JavaScript, Python and multiple other coding languages  Some flaws are that Mac Lacks windows only tools such as Visual studio, additionally Mac has Slower performance compared to Linux | Linux has many open-source development tools available such as eclipse; Linux also has many strong command line tools for developmental purposes. It is very secure and reliable. Linux is also known for running code quickly  However its flaws are that it Linux has a lack of native tools such as adobe, or visual studio. This leads to a smaller selection of tools to develop with | Windows contains visual studio; it also has native support of most languages. Additionally windows Has great options for debugging support. Finally windows contains large developer communities for insight.  Its flaws are that windows does not run as quickly as some other options as Windows is CPU and memory intensive. | Some mobile devices contain android studio; additionally mobile devices have extensive API’s that allow for notifications and other details. The large user base allows for easy insight.  However, mobile devices have many major flaws such as  Complex app store policies, strong licensing costs, and Lack of performance required to run larger projects. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

**Operating Platform**:

My final recommendation, given the pros and cons above, would be to develop on Windows. Although it is more expensive, you get a large number of resources that are very helpful for development, such as Microsoft tools and OneDrive. Additionally, it has a fantastic support system to help with development while also having the largest user base. Finally, it has access to many great development tools such as Visual Studio and Eclipse.

**Operating Systems Architectures**:

Windows uses hybrid kernel architecture; this architecture features both monolithic and microkernel designs, allowing it to keep speed while still running services in user space for stability. This structure would be helpful for developing *Draw It or Lose It* due to the fact that it is memory efficient. Additionally, this architecture is very compatible with other devices and software. Another benefit to this architecture is that it has support for multitasking, security, and scalability, which are all core concepts for development. It also has a fantastic ability to host and develop web applications.

**Storage Management**:

Windows runs storage on NTFS (New Technology File System). NTFS supports large files, file compression, encryption, and access to file permissions. It also has the ability to journal, which helps prevent data corruption in system failures. Windows has additional features to improve scalability and data recovery through Storage Spaces and its Resilient File System. All of these tools would be extremely helpful for both users and developers, allowing *Draw It or Lose It* to have a smooth and enjoyable experience.

**Memory Management**:

Windows runs on virtual memory management. This allows it to combine physical RAM and disk space to optimize performance and multitasking. This is a dynamic process ensuring *Draw It or Lose It* web servers and clients receive the resources they need while minimizing idle memory. Windows’ Memory Manager also handles paging, caching, and processes to prevent separate applications from interfering with one another. This alone will improve stability and response times, allowing the game to smoothly run and connect multiple players simultaneously. This allows *Draw It or Lose It* to expand its user base without worrying about impacting server stability and response times.

**Distributed Systems and Networks**:

In a Windows environment, *Draw It or Lose It* can use a distributed client-server architecture where the game logic and web applications run on a central Windows Server. Players then connect through web browsers or mobile devices via the internet. Communication would then be handled through RESTful APIs, allowing for real-time updates. Further communications, such as player interactions, would then be handled through network protocols such as HTTP and TCP/IP. If connectivity issues were to occur, then cloud-based backups such as Microsoft Azure would ensure continuous gameplay. This approach allows the game to scale while providing secure multiplayer experiences across any device a user chooses to play on.

**Security**:

Although Windows security can be more susceptible to attacks than other operating systems, additional steps such as firewall protection and encryption would allow the operating system to be more secure. User authentication and access control could be managed through Windows Active Directory and role-based permissions. This would ensure only authorized users could access sensitive information. Communication would be handled through HTTPS encryption, ensuring data leaks are prevented. Additionally, Windows Azure Security Center allows for monitoring, threat detection, and automatic defenses against vulnerabilities. This would allow Windows to protect users and ensure a safe and fun experience for all players.

All the above proves that Windows would be a great fit for *Draw It or Lose It.* From storage

management to security, Windows offers many developer and user tools to ensure that *Draw It or*

*Lose It* can be developed quickly, stress-free and on budget. I strongly recommend Windows

operating systems for The Gaming Room’s development of *Draw It or Lose It.*