

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.1

## 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 |
| --- | --- | --- | --- |
| 1.0 | March 19th, 2025 | Zaki Lodi | Initial changes and adjustments to Project files in response to the Client’s requests. |
| 1.1 | April 5th, 2025 | Zaki Lodi | Addressed the Server and Client-side attributes as well as the specific development tools that pertain to the various Operating Systems. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is looking to develop a web-based version of their game (Draw It or Lose It). The main design problem is the creation of a scalable system that can accommodate multiple concurrent games and players, whilst still maintaining unique identifiers and cross-platform performance. Given the cross-platform aspect there should be consistent testing within our timeline. In the initial stages of planning we should review previous and new requirements/ expectations to ensure the selection of the correct tech-stack, as well as making sure we properly allocate the necessary resources for the development team.

## Requirements

- Multiple-Team Support : The game should be able to accommodate more than one team for playability.

- Multiple Players per-Team : Teams can contain more than one player at a time.

- Unique Identifiers : The Game and Team names must be unique in order to prevent conflicts.

- Single-Game Instance : Only a single instance of the game can exist at any given time.

## [Design Constraints](#_2et92p0)

Some design constraints include:

* Cross-Platform Compatibility : The game has to function across various browsers and devices, so something like JavaScript implementation needs to be done carefully.
* Server Scalability : The Game system should be able to handle multiple games and users at the same time.
* Data Security : User login/ personal data should be made encrypted, and we should implement things like a secure login and session management.
* Modular Design : In case of future updates and additions to game libraries and features, the game should be made modular as to be able to adjust the architecture in chunks rather than overhauling the entire game every update.

## [Domain Model](#_8h2ehzxfam4o)

The UML Diagram below uses object-oriented principles to create a modular, scalable, and maintainable design for the “Draw It or Lose It” application, displaying the various classes and their relationships. The Game class has Composition with the Team class, the Team class has Composition with the Player class and Association with the Game class, and the Player class has Association with the Team class. We can see the encapsulation of the attributes and methods within each class (giving a clean interface). With the Game class having composition with the Team class, and Team with Player, we observe the “has-a” relationship, demonstrating the game’s requirement of multiple teams in a game and multiple players in a team. With the extensions of the Entity class, we can observe polymorphism and inheritance.

"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** | Mac in regards to the server-side for a web-based application comes with strong security and stability benefits. Since it’s based on UNIX we are able to take advantage of supported services like Apache and MySQL. Apache being especially important for hosting servers. Compared to other OS, Mac may offer less customization and be less compatible with certain software. That along with the consistent high costs of the hardware and software, there is often less community support for development needs. | Where Mac is typically very expensive, Linux finds itself on the other end of the spectrum being extremely cost-effective. Being entirely open-source and community-driven, there is compatibility with an abundance of tech and services like PHP, Apache, and MySQL. Also, with constant updates to most distributions, Linux has become known for little vulnerabilities to malware and cyberattacks. Though there are more Pro’s than Con’s, the Con’s are heavier. The main negative point being stability is contingent on your skill, and with a high learning-curve to the system there can often be massive bugs. Also, there is still a lack of compatibility with Windows and Mac specific applications. | Windows being one of the most used Operating Systems on earth offers seamless compatibility with a wide variety of tools and services like MS SQL Server, Visual Studio, and ASP.NET. Having decent stability, constant updates, as well as firewalls and DDoS protection, Windows is a tried and tested system in regards to safe online applications and functionality. Using Plesk we are able to create dynamic web applications along with data-driven websites. Some negatives to Windows are that besides being sometimes as costly as Mac options, it underperforms when it comes to the strengths of Linux. Such that it is often less flexible with open-sourced applications and services, and also may look sluggish compared to Linux with hosting high-traffic services. | The hosting of web-based applications on a mobile device is typically not feasible, and even if so it wouldn’t be able to scale up to this project’s requirements. It’s important to consider the limitations of hardware in regards to scalability, database operations, and high traffic volume. These negatives don’t make up for the portability and accessibility of modern mobile devices. |

| **Client Side** | As this application would be web-based, the huge negative of cost from the server-side wouldn’t apply here, most development would exist around the browser, so the need for local system-based software (proprietary and expensive) may not be necessary. It may take time to adjust to the macOS-specific browser behaviors (Safari). There is also the added quirk for Mac in making sure there is compliance with Apple’s design guidelines if the application is somehow integrated with macOS features. | With the overwhelming amount of support for all types of browsers and software, Linux should prove to be the easiest system for accessing the application over a browser. There may need to be a bit of compatibility testing across different distributions, but any developer familiar with Linux environments will have an easier time here than anywhere else. | Accessing the application over any dedicated browser would essentially be the same process as with a Mac, though there is more flexibility when it comes to customization and compatibility. | With many advanced IDE’s and tools for Android, these days it’s become easier to create apps that are able to seamlessly interact with web-based programs and databases. Besides a few UI differences, and the choice between an app running on IOS itself or from the browser, there’s not much complication here, though since many might be using iPhones it would be simpler to keep the application on the browser to be able to keep resources from the other platforms and save time. |
| --- | --- | --- | --- | --- |
| **Development Tools** | Java for back-end development and HTML/CSS/JavaScript (Vue.js). I’d use Eclipse if using Java, and Visual Studio for the web-development. | Same as with Mac. | Besides using C++ or C# instead of Java, everything else is the same as with Mac and Linux. | IDE’s like AppGameKit and GameMaker Studio would be good, and we could use JavaScript, C#, or Lua. |

## 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**: With prior experience using the Unity Game Engine for cross-platform projects, I would consider it ideal for the Draw It or Lose It game. Given the project requirement of future scalability and also ensuring the seamless additions of new game functionalities in the future, the Unity Engine is my first pick. The engine is able to be deployed to Windows, macOS, Linux, Android, iOS, and of course web browsers. Windows would be the base due to its great compatibility, developer support, and amazing integration/tools for Android specifically.
2. **Operating Systems Architectures**: As it pertains to some relevant issues stated above, the chosen architecture must be modular and layered. There would be a presentation layer that handles the user interface and also pulls user inputs and outputs, a logic layer that serves to manage game functionalities, rules, and processing, as well as the back-end data layer which acts as a persistent storage for the game. Unity is great for these requirements as it allows for efficient separation between game logic and platform-specific code.
3. **Storage Management**: As for storage, due to the project being cross-platform in nature it is ideal to use a cloud-native storage option. With lots of support for all the relevant platforms, “Open Saves” with Google Cloud is best suited for our needs. Essential tools and utilities such as a Unified API, built-in caching, and also great compatibility with other back-end storage systems like Cloud Storage, Firestore, and Memorystore.
4. **Memory Management**: To ensure Draw It or Lose It will run efficiently on devices with variable memory capacities, the Unity game Engine is a good fit alongside the virtual memory management that comes with most modern operating systems. Unity helps with object pooling and efficient asset loading and unloading for optimized performance and reduced memory leaks.
5. **Distributed Systems and Networks**: For cross-platform communication, a distributed architecture using RESTful APIs or gRPC over HTTPS is ideal. Google Cloud can be used for our needs, such that there is global reach and availability. The game server itself must be able to handle real-time game synchronization, session management, and managing outages with retry logic/ “graceful degradation”. Functionalities like load balancing and auto-scaling should be used to ensure reliability.
6. **Security**: The important methods to consider are the encryption of all data in transit and at rest (mainly considering TLS and SSL), solid authentication practices (OAuth and JWT), as well as maintaining consistent security audits and updates. Of course its also always important to ensure the best practices are being used during the development process in regards to secure coding and data handling. There is also the potential to utilize platform-specific security features, such as “Keychain” on iOS and “Keystore” on Android. Since we would be using Open Saves, there are multiple security features offered there as well, such as Monitoring and User Data Compliance features which can add an extra layer of protection.