

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 7/11/2023 | Michael Ladderbush | Original drafting of document |

**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 seeks to create a web application version of their Android only game *Draw It or Lose It.* Their needs for the application include only one instance of the game to be running at one time which can support multiple teams with multiple players. Each of these entities must be unique and allow for users to verify if their names are currently in use.

I would suggest we utilize the design patterns of inheritance and singleton in order to create a base entity class which can have a child game class restricted to a single instance. Our team and player classes will inherit from our entity class and utilize unique identifiers. By maintaining a list of teams within the game entity and a list of players within each team entity we can then utilize iterators to ensure uniqueness of each entity created.

## Requirements

* Restrict the application to only one game instance in memory at one time.
* Uniquely identifiable game, team and player entities.
* Game entity must allow for multiple team entities within.
* Team entity must allow for multiple player entities within.
* Functional port from Android to a web-based platform.

## [Design Constraints](#_2et92p0)

1. Using existing content from the Android application to develop a web-based environment will constrain design. Starting fresh will place a constraint on development time and budget.

The original code base which was created for the Android application can give developers a good foundation in understanding desired functionality. However, following the previous application can limit the polish and specificity of the final web-based application. Deciding on following previous data will require developers familiar with multiple platforms, increasing production cost. Starting fresh will extend development time but may reduce individual developer costs.

1. Web applications necessitate developing a website and finding a host server, both of which will place a constraint on development time and budget.

The need for a website will take development time and will also necessitate developers to have the ability to understand the interactions of both the front and back end of the application. The website will also require a host which needs to be accounted for in the budget and may affect design.

1. Ensuring the application will be playable on multiple platforms constrains design and budget.

Due to the web-application needing to be functional across multiple operating systems and browsers more development time will need to be dedicated to ensuring functionality in each environment. Developers will need to have an understanding of multiple platforms, how they will interact with one another and what will be required to accommodate the application. This will increase development costs and greatly determine the overall design of the application.

## [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 diagram below represents our application design. The application is stored in the com.gamingroom package and contains multiple classes. The Player, Team, and Game classes use inheritance in order to extend the properties of the Entity class to themselves. Each entity uses a constructor requiring a name and id.

The method to create a new instance of each class is held in the associated class. These classes have an associated relationship in which one GameService can have 0 or many Games, Games can have 0 or many Teams, and Teams can have 0 or many Players. When a method is called to create these entities, it uses its inherited constructor and is then stored inside the class. As an example, when a user wants to create a new player, they will use a previously created team entity to call “addPlayer()” with a name of their choosing. The method will then determine whether or not the given player name already exists in the Team’s list, “List<Player>” and if it does the user will be directed to choose a different name and if it does not that player will be created and added to the list. This same process is used in the Game to Team relationship. Determining uniqueness is done by using the Iterator design pattern, by looking over and comparing each list with the given input the user can determine whether the created entity will be a new entity or not.

The GameService class is associated with the Game class through a zero to many relationship, however the GameService class also restricts the number of Games allowed in memory by using the Singleton Design Pattern. The singleton is created by using a private static instance of our GameService constructor, which is in itself a parameter-less private constructor. This makes sure that a user cannot call or reach the GameService instance directly and must use the “getInstance()” method to access singular GameService with every call. Once a new game is added to the instance it will be the only game allowed to be in memory at one time.

Our GameService class and Entity class use getter methods in order to prevent users from mutating previously created entities, this improves the safety and portability of the code.

Lastly, our ProgramDriver class contains our main method and uses the SingletonTester class. Within the ProgramDriver we get an instance of GameService and name it “service” then we are able to call methods on service to create a game, which can then be used for teams and 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)

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 computers are known for their user-friendly environment and this extends to developers. Macs are stable, secure and are well supported. They can also tap into the Apple ecosystem to increase ease of development.  However, Macs are not tailor made for hosting servers. Macs lack many features of other operating systems that aid maintaining servers. The scalability using Macs is also a bottle-neck as they may not be able to handle high levels of traffic without relatively high-cost increases. | Linux is an open-source operating system this fosters a large and active community of developers, leading to continuous improvements, bug fixes, and security updates. Linux is known for its stability and reliability, making it a solid choice for hosting critical web applications. Linux is also usually available at no cost, significantly reducing hosting expenses for web applications.  However, Linux will require specific expertise as the command-line interface and different software installation methods may be less intuitive for those without prior Linux experience. While Linux supports a vast array of software, there might be cases where specific software is not readily available on Linux. Certain proprietary hardware components may not have optimal drivers for Linux, potentially leading to compatibility issues as well. | Windows has a widely recognized and user-friendly environment. And extensive software support. Web applications can easily integrate with Microsoft technologies when hosted on Windows, simplifying the integration process. Windows Server also includes Internet Information Services (IIS), a powerful web server that is well-integrated with the OS.  Weaknesses include the fact that Windows Server licenses can be more expensive than other operating systems. It also typically has a larger memory footprint and higher resource requirements compared to some other platforms. Windows has historically been more susceptible to certain types of security attacks. Proper configuration and regular updates are crucial to maintaining a secure hosting environment. | Mobile devices offer portability, allowing users to access web-based software applications from anywhere. Mobile device touch screens also provide a highly intuitive and interactive user interface. These traits can be helpful for developers who want to host and update their apps while traveling or developing in a non-permanent environment.  However, mobile devices often have less processing power, memory, and battery life compared to desktops or servers. Which will greatly impact the performance and capabilities of resource-intensive web applications. Mobile devices may experience intermittent connectivity issues, such as low network speeds or signal dropouts. Mobile platforms are also much more susceptible to security threats than other platforms.  Mobile devices run on different operating systems and come in various hardware configurations. Ensuring cross-platform compatibility would be a big strain on developers attempting to host an app. |
| **Client Side** | Apple hardware is known for being more costly than other options, and the more clients that require support will increase the cost of acquiring and maintaining many Mac products. Not only will purchasing and maintaining hardware increase cost and price of development but will also increase the amount of time necessary to develop the app.  Developers will need to understand the specifics of developing on Mac and because Apple’s yearly releases they will need to take possible changes into account. | Linux may require having access to a diverse range of hardware configurations to test your software thoroughly, increasing costs. However, the key advantage of Linux is that it is open-source, meaning that software costs will be greatly decreased compared to other platforms.  Multiple client types on Linux may lead to encountering more bugs and issues. Fixing and providing support for these issues might take additional time.  To support multiple clients on Linux, developers and system administrators need to be familiar with various Linux distributions and understand their differences. | Windows operating systems require licensing fees for each client device which can be a significant factor when supporting multiple clients. Ensuring compatibility across various client devices will require access to different hardware configurations for testing, further increasing costs. Windows also periodically releases updates and new versions. Ensuring that your software remains compatible with the latest updates and versions requires time to re-test applications.  Supporting multiple types of clients on Windows requires familiarity with various Windows versions but is thankfully very intuitive and has many tools to aid in development. Using Windows-specific development tools, such as Visual Studio or Windows APIs, can help develop software that works seamlessly across different Windows clients. | Supporting multiple types of mobile devices means accounting for different hardware capabilities and configurations increasing development and testing costs. Obtaining licenses for development tools and software development kits for each device can greatly increase costs as well. Distributing an app through official app stores like the Apple App Store and Google Play Store often involves paying fees for publishing and maintaining your app.  Supporting multiple platforms, such as iOS and Android, usually requires separate development efforts greatly increasing development time. Whenever new mobile devices or operating system updates are released, you will need to ensure your app remains compatible and optimized.  Supporting multiple mobile platforms demands expertise in the specific programming languages, development frameworks, and best practices for each platform, which places strain on development costs. |
| **Development Tools** | Swift is the primary programming language for macOS development. It's designed by Apple and is the recommended language for building native macOS applications. Objective-C is another language useful for maintaining and extending older Mac applications. C and C++ are also widely used in macOS development when working on low-level system components  Xcode is the official IDE provided by Apple for macOS and iOS development. Xcode supports both Swift and Objective-C and offers testing and debugging tools.  Homebrew is a package manager for macOS that allows developers to manage third-party software in your development environment. | C and C++ are widely used programming languages for developing software on Linux. Python is widely used for various purposes on Linux, including web development, scripting, automation, and system administration. Go is a relatively new language developed by Google. It is designed for performance, simplicity, and ease of use and is regularly used by Linux developers  Visual Studio Code is a highly popular code editor that supports a wide range of programming languages. It offers excellent support for Linux development. Eclipse is an open-source IDE with strong support for Java, C/C++, Python, and other languages.  Another important tool used with Linux is GCC, a collection of compilers for various programming languages, including C, C++, and Fortran. | C# is a popular programming language widely used for building Windows applications. C++ is a programming language that is commonly used for system-level programming and performance-critical applications on Windows. JavaScript is an essential language for web-based applications that run in web browsers on Windows.  Visual Studio is the flagship IDE from Microsoft. Eclipse is an open-source IDE that supports C++, Java, and various other languages. It is often used for cross-platform development on Windows.  .NET Framework: The .NET Framework is a software framework from Microsoft that is widely used for desktop and web-based applications on Windows. | HTML is the foundation of web development and is used to structure the content of web pages. Paired with CSS to control the presentation and layout of a web application. HTML is a very common language used for developing web applications. React Native is a popular open-source framework for building cross-platform mobile applications using JavaScript and React.  Android Studio is the official IDE for Android app development. It is built on IntelliJ IDEA and is equipped with tools specific to Android development.  Xcode is the official IDE for iOS app development. It offers a comprehensive set of tools for building, debugging, and testing iOS applications.  Microsoft's App Center is a unified cloud-based platform for building, testing, and deploying mobile apps. |

## 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**: My recommendation for the development of Draw It or Lose It would be to use the Windows operating system because of its ease of access, wide-range of server maintenance tools, and large user-base. The nature of Windows would allow the developers of Draw It or Lose It to focus on the game itself without having to worry about the learning curve of Linux and the cost investment of Mac. Windows is also a very popular platform making it easier for users to access the game. Windows also has access to many tools making web-development and managing of cloud-based services more developer friendly without the scalability issues present on Mac and the expertise required on Linux.
2. **Operating Systems Architectures**: The Windows operating system features a layered architecture. It operates in user mode and kernel mode, abstracting hardware through the Hardware Abstraction Layer (HAL). Device drivers facilitate communication with hardware, while system services offer functionalities like networking, file systems, and security. Developers can make use of the .NET apis and Microsoft also offers robust development environments that can aid in development as well.
3. **Storage Management**: I would highly recommend using web-based platforms for your game "Draw It or Lose It", which involves hosting the game on servers accessible through web browsers. This approach allows players to access the game without installing additional software, with the game's assets stored on cloud-based storage services. Player data and interactions can be managed through server-side databases, while web-based technologies like HTML, CSS, JavaScript, facilitate the user interface, game mechanics, and data management.
4. **Memory Management**: Windows manages its memory by employing techniques like virtual memory and caching to optimize performance. Virtual memory allows applications to use more memory than physically available by swapping data between RAM and disk storage. Caching involves storing frequently accessed data in RAM for faster retrieval. Windows uses techniques like demand paging, where data is loaded from disk only when needed, and memory compression, which compresses data in RAM to free up space. Furthermore, Windows aggressively releases memory occupied unnecessary processes to ensure efficient memory usage.
5. **Distributed Systems and Networks**: Enabling cross-platform communication for Draw It or Lose It will require utilizing microservices such as standardized APIs (RESTful). A central server that manages data synchronization, matchmaking, and interactions. Platform-specific clients connect to the server endpoint, which replies with the appropriate API, allowing for users to connect regardless of platform.
6. **Security**: To ensure robust security for Draw It or Lose It across platforms, I would recommend implementing a multi-layered approach. Methods like OAuth or JWT tokens to verify user identities on each platform, combined with encryption (SSL/TLS) for data transmission would allow the developers to ensure user data would not be accessible without proper verification. For web-based platforms, enforce HTTPS and adhere to OWASP security guidelines; in cloud services, configure firewalls and access controls. Regularly update and patch all components, follow secure coding practices, and educate users about strong passwords to safeguard their information.