

The Gaming Room

# **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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/21/24 | Andrew Willey | Added Summary and Design Constraints |
| 2.0 | 08/04/24 | Andrew Willey | Added Evaluation Table |
| 3.0 | 08/18/24 | Andrew Willey | Added Recommendations |

**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 looking to expand their Android based game, Draw It or Lost It, to a broader audience by having us develop a web-based game that serves multiple platforms. This will require us to develop the Android based game with the same mechanics and features to ensure a smooth transition for their audience. The staff at The Gaming Room does not have the necessary knowledge to develop such an environment for their transition so they have asked for our assistance in doing so.

## Requirements

* Cross Platform Accessibility – Web-based game must be accessible across multiple platforms.
* Multiple Teams- Has the ability to have one or more teams involved.
* Multiple Players- Each team must have multiple players assigned to it.
* Unique Naming- Game and team names must be unique.
* Single Instance- Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

* **Maintaining Features**:

The game development must include the same features that are already available on the “Draw It or Lose It” Android App. This is to ensure that users who already enjoy the game can smoothly transition and play the game on other platforms.

**Rationale**:

Maintaining these features is critical for user satisfaction and retention. Existing users should not experience any loss of functionality when switching to different platforms.

**Challenges**: Different systems have different screen resolutions which can affect the UI and display of certain game features. Different systems also have different input methods such as touch interactions on mobile and tablet and mouse and keyboard on home computers. We must ensure thorough testing on all platforms to ensure all features are usable.

* **Cross Platform Accessibility**:

The game must be able to run effectively on multiple different platform’s web browsers to reach a larger audience.

**Rationale**: Cross-platform accessibility will create more access for new users, effectively expanding the game’s reach, which is the goal of our client.

**Challenges**: It may be difficult to ensure that the game runs smoothly across all platforms due to varying levels of support from different web browsers and operating systems. This will require us to optimize our development to ensure the game runs efficiently no matter what platform the user is playing on. This will also require thorough testing across all platforms for performance capabilities.

* **Unique Team and Game names**:

Developers must ensure that team and game names are unique and allow users to check if the team names are already in use.

**Rationale**: It is essential that team and game names are unique to avoid confusion and errors when users attempt to play with other users.

**Challenges**: We need to make sure that our process of checking for previously used names is efficient so as not to slow down system performance. We should use lists and implement checks (like the iterator design pattern) for name uniqueness.

* **Single Instance of a Game**:

Only one instance of the game can exist in memory at any given time.

**Rationale**: Ensuring that there is only one instance of the game prevents inconsistencies and bugs when running the game.

**Challenges**: The challenge of ensuring that there is only one instance of the game at any given time can be solved by using the singleton design pattern. This will ensure that multiple instances of the game are not instantiated.

* **Web-based performance**:

We need to ensure that the game operates smoothly over various network conditions. We can accomplish this by using web sockets to maintain an open connection between the user and the Draw It or Lose It server.

**Rationale:** It is important to maintain a smooth gaming experience for the user no matter what their network condition is. This is important to maintain the user’s satisfaction and for our client to retain them as a continued user.

**Challenge:** It will be difficult to ensure smooth conditions across varying network conditions and speeds. We will implement strong web sockets to try and combat this issue and ensure user satisfaction. We can use tools to try and replicate varying network conditions to see how they impact the game’s performance.

## [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 Class:** Serves as the Parent class for Game, Team, and Player classes. It contains private attributes id and name as well as a private constructor, - Entity() and public constructor + Entity(id, name). It also contains public methods of getId(), getName(), and toString().
* **Game Class:** Represents a game instance that inherits from Entity and contains a private list Team, a public constructor + Game(id, name) as well as public methods of + addTeam(name) and toString().
* **Team Class:** Represents a team in the game that inherits from Entity and contains a private list Player, a public constructor Team(id, name), as well as public methods of addPlayer(name) and toString().
* **Player Class:** Represents a player on a team that inherits from Entity and contains no attributes but has a public constructor of Player(id, name) and a public method toString().
* **GameService Class:** Handles the creation of game instances. It contains a private list Game, and private attributes of nextGameId, nextPlayerId, nextTeamId and a private instance of GameService. It contains a private constructor of – GameService() and a public method of getInstance() to access the private instance of GameService. It also contains the following public methods: addGame(name), getGame(id), getGame(name), getGameCount(), getNextPlayerId(), and getNextTeamId().
* **SingletonTester Class:** Contains the method testSingleton() to test the singleton pattern used in development.

**ProgramDriver Class:** Contains the main method that drives the application’s execution.

Inheritance is represented by the Game, Team, and Player classes inheriting from the parent class of Entity. This allows Game, Team, and Player to share common functions which creates cleaner and more maintainable code. If a change needs to be made to shared functionality it only needs to be changed in the Entity class instead of in the Game, Team, and Player classes. Encapsulation is represented by using private attributes in all the classes and the public methods they contain to interact with them. This ensures that an object’s details are hidden from outside access and manipulation. Association is represented by the relationship between GameService and the Game, Team, and Player classes where each game contains teams, and each team contains players, and these are all maintained by the GameService class.

**"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** | **Characteristics**: Uses a Unix-based architecture, lots of built in applications / software that are great for creative projects. Offers macOS Server for hosting website-based apps and also supports Apache for web hosting.  **Advantages**: because macOS uses a Unix-based architecture it has a high level of security against malicious attacks.  **Disadvantages**: It is more costly than other operating platforms like Linux or Windows, it is also limited in its compatibility with other hardware / software. | **Characteristics**: Linux like macOS is a Unix-based system and it is open source which might cut down on cost of development. Offers server-based deployment through Nginx or Apache.  **Advantages**: high security, lower cost due to being open source, and is highly compatible with other hardware / software.  **Disadvantages**: Linux is a less popular operating system compared to macOS and Windows meaning it may be harder to find developers familiar with its command-line. | **Characteristics**: The most popular operating platform on the market, uses a hybrid kernel system. Can use Internet Information Services to host the web-based app.  **Advantages**: Because it is the most popular operating platform it has a high familiarity and comfortability with its users and developers. Has high compatibility with other hardware / software making it very versatile.  **Disadvantages**: Has higher security risks due to being targeted by malicious attackers. While not as costly as macOS it is more costly than Linux platforms. | **Characteristics**: Android, like Linux, is mainly open source, and has a variety of price point options. iOS is generally more costly but, like macOS is highly secure from attacks.  **Advantages:** Mobile operating systems are easily accessible allowing the web-based application to reach a broad audience. iOS has great security features while Android has many open-source options to lower development costs.  **Disadvantages**: iOS is generally more costly to develop on and Android like Windows is more susceptible to malicious attacks. |
| **Client Side** | The cost of developing on macOS platforms is going to be more costly than the other platforms and will require expertise in Swift / C. | The cost of developing in Linux will be lower than all the other platforms due to its open-source nature but will require much higher expertise and familiarity with Linux which may lead to longer development times. | The cost of developing on Windows will be less than that of macOS but higher than Linux due to licensing costs. Time should be less of an issue for development due to its large popularity / familiarity among users. It will require expertise in programming languages and development tools like Visual Studio. | Cost should be fairly low due to free access to development tools like Xcode for iOS and Android studio for Android. Time may be an issue if developing for both Android and iOS platforms as they will need to be developed on two different development tools. It will require expertise in Xcode and Android Studio. |
| **Development Tools** | The most common IDE for Mac is Xcode, which is free to download, and the relevant languages are Swift and Objective-C. A team that is proficient in Swift and Objective-C should be able to handle this project. | The most common IDEs for Linux are Visual Studio code (free community version but professional subscriptions vary from $499 to #5,999.00), Eclipse, and IntelliJ IDEA. Linux supports a wide range of popular languages like Java, Python, and C++. | The most popular IDE for web development on Windows is Visual Studio (free community version but professional subscriptions vary from $499 to #5,999.00) which supports languages like C#, Java, and C++. | This depends on the system being used. For Android the IDE of choice is Android Studio (free to use) supporting Kotlin, Java, NDK, and C++. For iOS Xcode (free to use) is used, supporting Swift and Objective-C. This will require multiple teams to handle each of the platforms (Android and iOS). |

## 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**:

The Windows operating system is an ideal choice for expanding Draw It or Lose It across multiple platforms. It offers support for a wide range of IDEs and programming languages, including C#, Java, and C++. Windows is also highly compatible with various software and hardware configurations. This will ensure easy integration with different development tools and third-party applications that may be used. Additionally, as the most popular OS on the market, it is familiar to most developers, reducing the need for additional training. Windows also offers different licensing options that can be customized to meet project needs, making it a more cost-effective choice for development.

1. **Operating Systems Architectures**:

The Windows OS uses a hybrid kernel architecture that combines the advantages of both monolithic and microkernels. It accomplishes this by placing some components in kernel space, like monolithic kernels, while other components exist in the user space, like microkernels. This design has two primary modes, Kernel Mode, which includes critical system components, and User Mode, where applications and certain system services operate with restricted access to system resources. This separation increases security by protecting the OS from user errors without compromising performance. Windows OS also optimizes performance and resource utilization through techniques like demand paging, which loads only the necessary pages of a program into memory, and virtual memory, which extends available RAM using disk space. These techniques help to improve system efficiency and performance.

**Storage Management**:

For Draw It or Lose It I would recommend using a serverless cloud-based architecture like Microsoft Azure. The use of Microsoft Azure would allow for unlimited storage scalability which would be ideal for a game that will have a large amount of players and game data across multiple platforms. Using serverless architecture would also mean that The Gaming Room wouldn’t require a dedicated team to manage its servers, reducing the cost of server maintenance. Microsoft Azure has a pay as you go option meaning The Gaming Room would only have to pay for resources used. They also have additional support options for security such as Microsoft Defender for cloud, and DDoS protection through Azure DDoS Protection. These options will help to protect user data and defend against malicious DDoS attacks.

1. **Memory Management**:

One of the big advantages of Windows memory management is its use of dynamic memory allocation using virtual memory. This is a memory management technique that allows the Windows OS to increase the amount of RAM available. Each process is given its own virtual address space, which appears as a contiguous block of memory, even though it is managed and mapped to physical RAM and disk storage by the OS. Through paging, Windows moves data between RAM and disk space, allowing the system to handle larger data sets than just physical RAM would permit. This technique of dynamic memory allocation will allow Draw It or Lose It to load and display images and game data only when needed, effectively reducing memory usage.

1. **Distributed Systems and Networks**:

Draw It or Lose It will use RESTful APIs to implement a client server design pattern that will allow communication between the client (web application) and the server. The server side, whether it be a centralized server or a cloud-based server like Microsoft Azure, will be responsible for the game state, player and team information, user authentication / authorization (using role-based access control), and the delivery of the two hundred different high-definition images. The client will be responsible for the user interface and processing user input, sending requests to the server via defined API endpoints to retrieve data and update the game accordingly. It is crucial that The Gaming Room choose a strong network system to maintain connections between clients and server. The web application relies on the server for functions such as updating the game state, authenticating users, and delivering content like images and player or team information. A cloud-based server solution like Microsoft Azure is particularly useful in this scenario, as it offers large scalability and can efficiently handle high volumes of requests, regardless of the number of active players. Microsoft Azure also has several useful tools to help mitigate outages and connectivity issues such as Azure Traffic Manager (helps to manage server traffic), Azure Site Recovery (replicates workloads to a secondary region, allowing you to fail over and resume operations quickly), and Azure Backup (provides automated backups which helps mitigate the impact of connectivity issues).

**Security**:

Windows comes with built in Windows Security that includes multiple sign-in options like Windows Hello PIN, security key, password, and picture password. It also has something called dynamic lock which pairs your PC with other devices to know when you’re away and locks your PC if you go out of range. These options are great for protecting against unauthorized access to your PC at a physical level. Windows also comes with a Firewall to monitor and protect against unwanted network connections. Windows Defender Antivirus is also included and can help detect and protect against malicious malware attacks. These are all great security options to help mitigate malicious attacks but, there are also security techniques that can be implemented to further protect user information. This includes RBAC (Role Based Access Control) which will only permit access to those with the required role, effectively limiting access to sensitive information that is not required for users to complete their work. If The Gaming Room does decide to use Microsoft Azure, there are options as I’ve listed above for increased security like Microsoft Defender for Cloud and even Microsoft Defender External Attack Surface Management that helps to identify potential risks in your system.