

# **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 | 08/18/2024 | Dylan Dunagan | Added company problems, their requirements, technical and business constraints, explained the UML diagram, and explained the pros and cons of using different operation systems for the desired game. Added an evaluation from the server and client sides, as well as required developmental tools. Added personal recommendation for development. |

**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 Company, The Gaming Room, wants to develop an app for one of their already existing games, Draw It or Lose It. While the web-based game is already available on Android devices, The Gaming Room wants the game available on multiple other platforms. The game consists of multiple teams who will guess what is being drawn over a timed period. If one team does not correctly guess the image, then other teams could guess the answer instead.

## Requirements

The technical requirements are that The Gaming Room would like one or more teams, which could have multiple players in each, to have the ability to play at a time. They would like the teams to be unique to the point that they are able to check whether teams’ names or profiles already exist. Additionally, they would like only one instance of the game to exist at a time. Meaning, the same game instance will not be played twice at the same time. Another technical requirement that could also be a business requirement is to have the game web-based, and available on multiple platforms.

## [Design Constraints](#_2et92p0)

The first design constraint is to have unique team/player names. This will require a list to be created and saved as players make their teams. The program will then be able to look through the list as new teams are created to verify if that new team can be created. Similarly, the next design constraint will be maintaining only one game instance in existence at a time. The program will again have to make a list of the game instances so that the same instance will not be able to be played while already in use. Next will be allowing the game to be on multiple platforms which would potentially require different coding languages so that the program can run on those different devices.

## [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 first class is the Entity class. This class is where the program will get the game identification and team names to the pass along to the other classes. The Player, Team, and Game classes derive from the Entity class. The Player class is where the program creates a player identification and passes it to the Team class. The Team class then verifies if a team name is already in existence or creates a new team and adds players to those teams. Then the Game class checks for a game that is already in existence or adds teams to a new game. Everything then gets passed on to the GameService class where it creates the game instance to be played and allows for the game instance to be saved in its own game list. The ability to verify that only one team, game, or game instance is in existence is because of the SingletonTester and ProgramDriver classes, which test to make sure that multiples of the same name cannot exist.

**"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** | Integrates very well with other macOS devices but may provide hardware limitations regarding reliability and performance. | Known for stability and reliability with servers, having a long uptime. Linux can handle efficiently handle high loads of information. Hardware can reduce compatibility with Windows and macOS. | Integrates with other Windows products very well. Windows Server can help with security management due to ease of authentication, but Windows requires more effort due to being a common target for security vulnerabilities. | Have limited resources such as processing, memory, and storage power. Can enhance functionality due to built-in sensors, like GPS and is very convenient for users to use. |
| **Client Side** | Requires Apple hardware and software which can be very expensive. Software testing requires compatibility with the various versions of macOS. Unless familiar with macOS, developers require expertise in specific parts of macOS. MacOS gives users a specific user experience that must be adhered to for proper user satisfaction. | Linux is open source making it a much cheaper option compared to macOS and Windows, but other costs could exist with using different licenses or software tools. Supports an abundance of programming languages, allowing for a shorter development time. Requires expertise with Linux functions and capabilities. Linux does provide a large community which provides resources with development. | Hardware is less expensive than macOS devices, but a lot of Windows development relies on different software that could require licenses or subscriptions. Also supports a variety of programming languages which shortens development time. Testing only requires compatibility with Windows 10 and 11 rather than the variety of macOS systems. Requires expertise in Windows technologies, how Windows integrates, and knowledge of Windows security. | Typically requires specialized tools specific to each mobile operating system. Requires testing on multiple mobile devices, such as various phones and tablets, and different screen sizes and hardware. Can involve paying app store fees. Can be time consuming due to the multiple specific operating systems. Requires expertise in mobile specific languages, as well as optimization too allow for efficiency with app performance and batter life. |
| **Development Tools** | MacOS uses Swift and Objective-C as their programming languages for all devices. Xcode is what macOS uses for development which provides tools, simulators, and analyzers to develop for all macOS devices. These are essential to developing on macOS systems but can not be used on other systems. | Linux supports the programing languages C, C++, Python, Java, and Go, all incredibly common programming languages. Linux supports the IDE’s Visual Studio, Eclipse, IntelliJ IDEA, EMAC, and Vim. These provide a variety or paths to take when developing software and applications. | Supports C#, C++, .NET, Java, and Python, also all very common programming languages. Also supports Visual Studio and Eclipse. Allows for creating software, web services, and games. | Requires Swift, Objective-C, Java, Kotlin, C#, JavaScript, Dart and more, due to the abundance and variety of mobile devices and the operating systems on them. Requires Xcode for iOS, Android Studio and IntelliJ IDEA for Android products, and Visual Studio for Microsoft products. |

## 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**: I recommend using Windows as the operating system due to the reduced time for development as well as the accessibility for the consumer. Windows supports a variety of programming languages that are compatible with other operating systems. The cost and abundance of physical hardware will also allow for Draw It or Lose It to expand over time. This expansion will allow for easy and well-done upgrades on the server side as well as on the consumer’s side.
2. **Operating Systems Architectures**: Windows uses a variety of programming languages. Windows is also a 64-bit system allowing for multitasking. Windows is an operating system for phones, tablets, as well as video game consoles allowing for Draw It or Lose It to be played on many platforms that are cheaper for the consumer to acquire. Windows is leading the race with gaming features and has the most capabilities as well as compatibility with video games.
3. **Storage Management**: A commonly used storage system in Windows is Storage Spaces. It is ideal for file servers and data storage by helping create storage pools while maintaining data resiliency. Storage Spaces also allow for scalability and flexibility as storage requirements grow. As a web-based program, this could be a viable option for a storage system. As mentioned before, as more storage needs presents itself, it is incredibly easy to add more storage to a Windows device.
4. **Memory Management**: Windows will put background applications to sleep to free up memory for the application in use. Windows also allows the user to specify their memory priorities for a better consumer experience. Windows compresses memory and stores it in its RAM memory to save memory space on local discs and drives. All of this helps to minimize latency, maintain responsiveness, and optimizes memory allocation, which is essential for any kind of video game.
5. **Distributed Systems and Networks**: For Draw It or Lose It to be a web-based game, it would need a game server to host game sessions, manage players, and keep the game in sync between teams. This would depend on the server and the consumer having stabile network connections. There could also be Peer-to-Peer networking option, which would allow for multiplayer interactions without a server or a connection (perfect for outages), but the game would be local to that on device or potentially be able to be played through a Bluetooth scenario.
6. **Security**: One security tool all systems are capable of is account and password management. This relies on the consumer to create strong passwords for their accounts. Windows uses BitLocker Drive Encryption than can encrypt the entirety of a drive which protects user information by preventing unauthorized access. Windows supports HTTPS type websites which is a way to encrypt information on the website, which is very useful for a web-based game. Windows also provides updates almost weekly to consistently update security needs.