

Draw It or Lose It

# **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 | 07/20/25 | Andrew DePasquale | Updated key sections: Summary, Development Requirements, Constraints and Recommendations. |
| 2.0 | 08/03/25 | Andrew DePasquale | Updated key section: Development Requirements |
| 3.0 | 08/17/25 | Andrew DePasquale | Updated key section: 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 client is looking to transform their Android only game, Draw It or Lose It, into a multi-platform application. They aim to release it on different web based platforms such as: Mac, Linux, and Windows. They have reached out to us in hopes of helping expedite the development of this web-based version and help set up the environment for it. Regarding the game, each team name along with game needs to be unique, and the game needs to only have one instance at a time. Each team will also be assigned multiple players, and there can be one or more than one team during a game.

## Requirements

* *Only one game instance exists at a time*
* *Each team must have multiple players*
* *Multiple teams can participate in each game*
* *Unique names required for both teams and games*
* *Entities such as game, team and player should have unique ID as well as name*
* *Singleton design for GameService class*
* *Iterator for finding/managing games, teams and players*

## [Design Constraints](#_2et92p0)

* Application needs to support Windows, Linux, Mac, and Android OS.
* Only one instance of the game may exist in the memory, needing Singleton implementation.
* System needs to have name uniqueness for entities, requiring that we utilize iterators to check memory for similarities and require creation of new ones if name is already in use(regarding games, teams, and players)

## [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 Entity class acts as the superclass for Game, Team and Player. This class contains the common attributes of id and name. Inheritance is shown by this class allowing the subclasses Game, Team and Player to reuse these attributes, essentially Game, Team and Player classes extend from the Entity class. The Entity class also promotes encapsulation with it having private fields and public getter methods. Abstraction is also shown through instances like the GameService class providing public methods like addGame() and getGame() without the user needing to know how these are stored. The last principle of polymorphism is exemplified by having different types of objects such as: Game, Team, and Player objects created through the subclasses that then become different instances of the Entity superclass.

The Game class contains a list of Team objects and demonstrates a one to many relationship between the Team class. The Game class also implements the addTeam() method to add new teams. The Team class contains a list of Player objects and demonstrates a one to many relationship between the Player class. The Team class also implements the addPlayer() method to add new players. The Player class does not manage any internal lists like the Game and Team classes, but does represent an individual participant within a team. The GameService class implements the Singleton pattern to ensure there is one instance existing at a time along with utilizing iterators to ensure uniqueness with the names of the games, teams and players. The GameService class also stores the list of games and manages the IDs for games, teams and players. The ProgramDriver class contains the main() method which is the entry point for the program and uses the SingletonTester class to verify the Singleton pattern is used properly. Lastly, the SingletonTester class contains the testSingleton() method to ensure only one instance of GameService is used.

**"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** | Stable and very user friendly, but also very closed off and restrictive making it harder to run many programs. Licensing is generally cheaper the Microsoft but must be run on a Apple owned OS which can lead to difficulties. | Fast, secure and stable, but can have compatibility issues with certain software or programs, similar to Mac. It is typically free to use and preferred by most cloud providers such as: Amazon Web Services and Google Cloud Platform. | Widely compatible with many software and programs making it easy to use, but comes with more security issues compared to Mac and Linux. Licensing can be $1000 or more which can be quite expensive. | Very accessible and convenient, but have limited power and storage compared to the other operating systems making it harder to run programs and servers, making other OS more favorable for such tasks. |
| **Client Side** | Cost can be higher due to the need for apple specific tools, which can also make the time of development longer as well. Developers will need extra expertise with the Mac OS specifically and have a greater understanding of apple products and programs. Safari is the main web browser for Mac OS so we would need to do testing during development to ensure it runs smoothly and correctly on it. | Cost is generally low due to Linux software being open sourced and free. Development time can be faster due to this availability and the flexibility of the environment, this can make the expertise side for developers easier, they just need to understand Linux commands. Linux users typically utilize Firefox or Google Chrome, so testing on both platforms would be necessary during development to ensure a smooth connection on them. | Cost may vary depending on the licenses of certain tools and programs that may be needed, the time of development would be moderate since Windows is one of the most widely used operating systems, this can make the expertise side easy as well depending on how well developers understand Windows specific technology. For Windows, we would need to test Firefox and Google Chrome like with Linux, but Microsoft Edge as well since it is a main browsing platform on Windows OS. | Cost may be higher due to the need to develop multiple platforms iOS(iPhone) and Android along with possibly needing developer licenses. Development time might be longer due to needing the program to function on specific screen sizes and operating systems. Expertise is crucial since developers need to understand limitations and specific coding languages used for mobile app development. During development we would need to test for different screen sizes and ensure it performs well, along with the main browsers for each. iOS (iPhone) utilizes Safari, and Android utilizes Google Chrome as their respective browsing platforms. |
| **Development Tools** | The programming language Swift is commonly used for Mac development. The main tool is Xcode, which helps build, test and deploy Mac applications. Xcode costs about $99 for a year subscription which is on the cheaper side. For a development team side of things, it would need to be specified with people well versed in apple products and how they function with a Mac specific team rolling out the program. | Many programming languages can be utilized for Linux including: Python, C and Java. A common tool for Linux is Eclipse which helps developers build, test and deploy within Linux. The coding languages for Linux remain free to use which makes it an excellent option, along with Eclipse also being free bypassing the need to buy licenses like other OS. For development on this OS it would be more streamlined and easy with these coding languages being used by most developers. | Common programming languages utilized on Windows are C# and C++. The most common tool for developers to build, test and deploy on Windows is Visual Studio. Visual Studio costs $499 which is by far the most expensive license making it a hard option despite being the most utilized OS. Windows is commonly used by most and as long as the development team is well versed in .NET and Visual Studio, then it should be very efficient and manageable developing on this OS. | Common programming languages for mobile devices are Swift for iOS or Kotlin for Android. Common tools for developers to build, test and deploy on mobile devices are Xcode for iOS and Android Studio for Android. Xcode again is $99 for the year which is cheaper than Windows and Visual Studio, but there are limitations with mobile OS. Android Studio though is free to use which could make it a great option. We may need multiple teams for this OS, with one working on Android and the other working on iOS for iPhone. We might be able to get away with one team if we find a way to cross platform but most likely will utilize more than one for this job. |

## 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**: Windows Server integrated with Microsoft Azure would be the recommended operating platform for Draw It or Lose It across multiple environments. Microsoft Azure integration would allow for scalability, equal loading amongst many users simultaneously, and high availability essentially making it accessible at all times. Windows Server supports a wide range of frameworks and tools such as .NET and RESTful APIs, making it highly compatible with the Android version that already exists along with cross-platform development. Windows also benefits from a large development community which could accelerate development and allow for quick resolutions to any issues that arise.
2. **Operating Systems Architectures**: Windows Server utilizes the Universal Windows Platform (UWP) architecture, that allows applications to run on multiple devices such as: PC, tablets and mobile devices. UWP supports a wide range of APIs including Win32 and .NET, which provides a very flexible environment. This architecture makes it easier to develop apps that work on different screen sizes and device types.
3. **Storage Management**: Azure Blob Storage is a reliable and compatible for Windows development, as it is reliable, scalable and cost efficient object storage. It is great with large unstructured data such as: game state, user assets and media files. Azure Blob Storage is also compatible with other Microsoft and Windows services making it a good choice for the Draw It or Lose It game.
4. **Memory Management**: Windows uses virtual memory to help manage RAM and run multiple programs at once. This will allow the Draw It or Lose It game to utilize memory efficiently, helping it run smoothly and avoid crashing while also improving multitasking. Windows also uses demand paging which means only the necessary parts of the program are loaded when needed, which helps conserve RAM.
5. **Distributed Systems and Networks**: To let Draw It or Lose It work across different platforms, using a distributed system with web services or REST APIs is a good idea. These let devices on different systems share data through the internet or local network. A centralized backend hosted on Microsoft Azure can serve as the core for this distributed system managing user authentication, the game logic, data storage and synchronization allowing the communication between clients. These clients would communicate with the backend through standardized RESTful API calls, ensuring platform independence and consistent behavior. The network should also be built to handle problems like connection issues by including retry logic along with data syncing.
6. **Security**: Windows already comes with security features such as Windows Defender that helps to protect files and devices. Utilizing tools like HTTPS for communication along with user logins and permissions will help to protect personal data. If the game ends up using Azure, it could take advantage of advanced security tools like threat detection and multi-factor login, helping further protect user data and privacy.