

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 03/17/23 | Lyndsey Quintana Hernandez | Completed the executive summary, design constraints, and the domain model |
| 2.0 | 04/01/23 | Lyndsey Quintana Hernandez | Updated “Development Requirements” table. |

**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 develop a web-based version of their game “Draw It or Lose It.” The game will need the ability to have multiple teams, multiple players per a team, require unique team names, and only allow one instance of the game open at a time.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* Allow only one game instance in memory through unique identifiers.
* Ensure team names are unique.
* Be compatible with multiple platforms and Operating Systems.
* Allow multiple teams and players per a team.

## [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 is the parent class which allows Game, Team, and Player to inherit attributes and functions from it. Inheriting allows attributes and functions to only be coded once so all subclasses/children classes can pull these attributes. GameService, Game, Team, and Player all share the same multiplicity between. This allows us to add many players to each team, many teams to each game, many games to the game service. The ProgramDriver calls on the SingletonTester to determine if only a single occurrence of the game is open and is an example of abstraction as it limits the view of the background processes and only tests if once occurrence is open. Polymorphism is also witnessed through Team, Game, and Player. They pull from the parent class and then also run through their own 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** | Mac tends to be more expensive. It has high stability and adaptable commands.  Advantages: Great security, smooth web code through Apache, easier dual OS setup than Windows.  Disadvantages: Not easily scalable, Less hardware and software options than windows, higher cost, and less of a market percentage. | Linux open source which makes it more cost effective (free).  Advantages: shares similar advantages to Mac. High stability, high security, high compatibility except with windows, convenient access.  Disadvantages: Compatibility issues with Windows which has the highest market percentage. A large learning curve. Migration issues almost guaranteed to happen. | The most market dominant of OS behind Android. Medium cost of implementation  Advantages: Largest amount of software and hardware available. Easy UI, convenient access to tools.  Disadvantages: bigger target for malicious actors, frequent QA issues (bugs). Not able to run on Mac or Linux devices. Middle of the road costs. | The most popular OS and platform with high portability. Should use cloud hosting.  Advantages: Cost effective for upgrades and easy implementation.  Disadvantages: compatibility for each type of OS, connection issues, hardware limitations, not scalable (only small task driven), and security concerns. |
| **Client Side** | Moderately easy to use. UI is more streamlined and similar regardless of mac device used. This cause lower cost and time. | Free due to open sourcing. UI is difficult to use for beginners. High collaboration allows QA to be higher. Higher time and expertise. | Not easily used on other OS and platforms. Moderate cost associated. Low expertise due to easy UI. Lower time due to easy UI. | No scalability, limited offline function. Higher expertise to locate numerous functions. Cost efficient and lower time constraints. |
| **Development Tools** | Mac is powered by C programming language and uses Swift but is compatible with all programming languages. There are many compatible IDEs for mac: Microsoft Visual Studio, Android Studio, Pycharm, Xcode, Netbeans all have Mac compatible versions. | Linux supports C, Python, C++, Java, Perl and many other languages. The best compatible IDEs to use are Sublime, Atom, Eclipse, or Bluefish. | Windows would utilize HTML, Javascript, and C# programming languages. The most popular IDE is Microsoft Visual Studios. | Mobile devices would use a hybrid programming language for best platform compatibility such as, C# or javascript. IDEs available are Android studio, atom, Sublime. |

## 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**: Considering the required degree of expertise and the relatively low cost, I suggest that The Gaming Room begin the project with Window OS. Window OS also utilizes an multitude of practical IDEs that will facilitate the process and make it more efficient.
2. **Operating Systems Architectures**: Windows architecture permits applications to utilize the platform's kernel processes without influencing them directly. In other words, programs can take advantage of the capabilities of Windows to set up a graphical user interface (GUI), gain memory access, and perform other essential functions without impacting the processes that make the OS function.
3. **Storage Management**: Disk Management is integrated into the Windows OS.   Additionally, Windows includes Disk Cleanup and Storage Sense. Disk Management is a Windows system feature that is primarily used for advanced storage functions, whereas Disk Cleanup and Storage Sense are used to manage the system's storage by removing unwanted files.
4. **Memory Management**: Memory Management is an integrated Windows system tool.  However, there would need to be a database created for all of the images and it would need to allow the application to readily access them.
5. **Distributed Systems and Networks**: In this case, a client-server distribution system would be best, as each client application will rely on the game's singular server application, allowing each application to be tailored to the client's system strengths. Additionally, a robust server network would be required because this game relies on multiple customers being able to connect to a central server to play a single game.
6. **Security**: Windows has its own security software built in. Given that the game will be interacting with sensitive information, other software should be considered if the organization is willing to increase the security posture of the entire system for an increased charge. McAfee or Norton products are examples of some programs that will safeguard them from harmful interactions with malware or viruses that pose a threat to our users' and company's sensitive data.