

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 | 03/18/23 | Bobby Rust | Initial commit |

**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)

Draw It or Lose It is a game based on a 1980s television game and is currently only available on Android devices. Our client, The Gaming Room, would like to develop a web-based version of the game in order to run on many different platforms. The game renders from a large collection of drawings (images) which act as clues. The teams use these clues to compete and guess what is being drawn. One game consists of four one-minute rounds. The drawings of clues are rendered at a linear rate for the first 30 seconds of each round. It is turn-based, as one team gets to guess at time. If the guessing team does not correctly guess the puzzle before the time is up, the non-guessing teams each get 15 seconds to make a single guess. The scope of this document is constrained to software design decisions only.

## Requirements

1. Only one game instance can exist in memory at once.
2. A single game can have one or more teams.
3. Each team will have multiple players.
4. The game and team names must be unique.

## [Design Constraints](#_2et92p0)

Only one game instance of the game can exist in memory at once, so a singleton design pattern is used to ensure that this requirement is met. A single game consists of one or more teams, so there is a zero-to-many relationship between a game and teams. There is also a zero-to-many relationship between a team and players. The game and team names must be unique so an iterator design pattern is used to check and ensure that a name does not exist before creating a new one.

## [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)

* ProgramDriver: This class uses the SingletonTester class and contains the program’s main() method.
* SingletonTester: This class has a single member function, testSingleton(), which is used to ensure that there is only one instance of Game in memory at once and can be ran repeatedly as new code is incrementally added to ensure that the tests still pass.
* Entity: The base class which is the superclass to all other classes except ProgramDriver and SingletonTester. It contains member variables and functions that are to be implemented by all subclasses. This ensures that the children each follow the correct blueprint while also not unnecessarily repeating the same code in each class.
* GameService: Has a zero-to-many relationship with the Game class, so a GameService instance can have any number of Games associated with it. Its member variables are private so they cannot be called by anything other than GameService. They are static, meaning they belong to the class itself rather than an instance of the class. It implements the singleton design pattern so that only one instance of GameService can exist in memory at once.
* Game: Has a zero-to-many relationship with the Team class, so a Game has zero or more teams associated with it. It inherits from Entity and overrides Entity’s toString method.
* Team: Has a zero-to-many relationship with the Player class, meaning a team can reference zero or more players. It inherits from Entity and overrides Entity’s toString method. A method is defined to add players to the team.
* Player: Inherits from Entity and overrides Entity’s toString method.

**"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 is Unix based, giving it some of the features of the Linux operating system. Mac is easy to use with a built-in package manager, meaning it is easy to install and manage various software packages. Mac has a robust security system, making it the most secure platform compared to Windows and Linux. It is more expensive than Windows and Linux and has a smaller market share than Windows, meaning less software will be available. | Linux is open-source, making it highly customizable and flexible. Its command line interface gives its server administrators more control. This is also a weakness of the operating system, however, as it requires a well-trained server administrator to set up an environment. It is stable, reliable, and well-tested, as many critical systems have their servers hosted on Linux. | Windows is popular and has the highest market share, and it has the greatest amount of software available on it as most programs are built with a Windows-first mentality. It has a wide support system as most people use Windows. It is user friendly like Mac, and it is closed source unlike Linux, so it is not as customizable. Windows has great compatibility with software. | Mobile devices have limited processing power and storage, making them less ideal for hosting web-based applications. They are portable and have built-in touch screen support, making them user friendly. They have less compatibility with software than a desktop operating system environment. Mobile devices may be sufficient for small scale project, but not larger projects. |
| **Client Side** | Mac may require higher time and cost to support. Mac has less compatibility and requires more specialized knowledge than Windows. | Linux is similar to Mac in that it can require more time and specialized knowledge, but the cost can be lower than Mac. Linux offers customizability and flexibility making it easier to develop on in some cases. | Windows has the greatest number of developers trained for it. The cost is thus lower. The time it takes to develop can be lower as there is greater support for this platform. | Mobile devices require specialized knowledge as they are not as compatible with software. However, recently there have been development tools that allow software to be developed and run on multiple Mobile platforms without having to develop separately for each. |
| **Development Tools** | Xcode is a popular IDE available for Mac to build web-applications. Swift is a popular programming language for Mac. Vapor is a Swift framework for building web apps, which can simplify development of web apps and increase productivity. | Similar languages and frameworks can be used between Windows and Linux for web development, such as Python, Java, and Node.js, which is a JavaScript library. Visual studio code can run on both Windows and Linux. | Popular languages for building web applications for Windows is JavaScript. Another popular language for building web apps is C#, which provides a ASP.NET framework for building web apps. A popular framework for building web applications on Windows is React.js. | React Native is a popular framework for building mobile applications. It can be run on multiple Mobile platforms, removing the need to develop multiple separate projects for each platform. |

## 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**: An operating platform that I recommend for Draw It or Lose It is Linux. Linux is a very popular choice for hosting servers. It is open source, giving it great customizability. There are also many Linux system administrators available for hire as it is an in-demand role.
2. **Operating Systems Architectures**: Linux uses a monolithic kernel, which means that all of its components are working in kernel space, making it very performant. As previously stated, it is also open-source and modular, making it easy to enhance the operating system’s capabilities as desired. Linux is also a great choice due to its security, trumping both Windows and Mac in this department, keeping the servers safe and secure from breaches.
3. **Storage Management**: Storage management involves managing the allocation and deallocation of storage resources. The Linux storage management system I would recommend is XFS. It is scalable and performant, making it a good fit for the Draw It or Lose It platform. XFS offers journaling for metadata operations, which protects the system in case of a server or system crash.
4. **Memory Management**: Memory allocation is the process of giving memory to applications as needed and freeing that memory for use by other applications when it is no longer needed. Linux uses virtual memory, which increases efficiency by providing the ability for applications to use more memory than is physically available. Linux also utilizes demand paging, which is the process of loading into memory only the parts of an application that are needed, leaving more memory free for other uses and reducing system load.
5. **Distributed Systems and Networks**: With clients on various devices, communication between clients could be achieved by using a client-server architecture. In this architecture, the server can be running on Linux, and various clients on different devices can communicate with each other through the server. The client sends requests to the server to perform various tasks such as updating data. The server fulfills that request by updating the data in its storage and sends back the data to the client if necessary. Multiple servers could be hosted in various locations to ensure the consistency of the client’s experience. The client would connect to the nearest server, or if that server was overwhelmed with usage, the next closest server could be used. This is load-balancing, which is a technique used to keep the systems stable during peak usage.
6. **Security**: Linux offers a variety of security features such as user access controls, firewalls, and encryption. User authentication should be implemented on the server to confirm that the client sending the request is valid. This prevents malicious clients from sending a request as another user to access sensitive data. Access controls prevent an individual from gaining access to or altering sensitive data on the server. The firewall will only allow authorized network traffic from reaching the private network.