

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Document** **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 | 09/12/2023 | Robert Umland | Original Version |
| 2.0 | 10/01/2023 | Robert Umland | Updated Evaluation |
| 3.0 | 10/09/2023 | Robert Umland | Updated Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room, a software company, produces an Android-only game – Draw It or Lose it. This is a team-based game that can have many players per game. The client would like to expand it to be a web-based game that can have many instances of the game running at any one time. Each instance needs to be unique. The challenge is in setting up the new environment.

## Requirements

1. A game will have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

1. Web-based: The new version of the game will be accessible online to anyone with an internet connection and an appropriate web browser.
2. Interface: The game will be accessible to computer users as well as mobile device users. Consideration should be given to the different requirements for each version.
3. Unique Instance: Games must have unique team and player names.
4. Security: Since the game will now be web-based, strong consideration needs to be given to the security of the site and any personal information provided by the users.

## [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)

Class relationships:

1. Entity is the superclass for Game, Team, and Player classes.
2. GameService, Game, Team and Player classes are related through association/aggregation to one another.
3. ProgramDriver class has a dependency relationship with the Singleton class.

OOP Principles

1. Inheritance: The Entity class is the super class for Game, Team, and Player classes. These subclasses inherit the attributes and methods from the Entity Class.
2. Encapsulation: All the classes have attributes – and some have methods – that are private. The attributes are kept from direct manipulation by other classes and controlled only by the methods defined in that specific class (setters and getters).
3. Association/Aggregation: Many of the classes share a relationship through their objects, which is association. GameService, Game, Team, and Player all show association with the List attribute. And as each of these is a one-way relationship, the association is a specialized form known as aggregation.
4. Dependency: The ProgramDriver class depends on the SingletonTester class for testing that the program is written in the Singleton pattern. As shown with the <<uses>> text in the UML. This is known as dependency.
5. Singleton Pattern: To ensure that only one instance of a named game is running, the code is written such that the constructor in GameService is private and the getInstance() method is static. This is part of the Singleton design pattern.

**"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** | MacOS is a possible option for server side. The OS is Unix based which is a familiar coding environment, it is a reliable and stable host option and supports multiple web hosting software options like Apache or Kestrel (which is cross-platform).  While an option, there are several issues using MacOS as a server. MacOS is expected to run on Apple hardware. Apple offers only small server deployment support and is difficult to scale for larger deployments. There are license fees and higher hardware costs. | Linux is an excellent option for server side of webhosting. There are many free distributions of Linux with some that are designed for large-scale server duties. There is a strong community of developers constantly improving and optimizing the OS for server use. There are various web hosting choices such as Nginx and Caddy Webserver. While versatile, Linux can be a steep learning curve and requires more expertise for setup and maintenance. | Windows is a good option for hosting a web-based application. It is a very familiar environment for many developers. There is excellent support for .NET-based applications and robust support in general. Microsoft has a specific Windows Server OS developed expressly for server work.  While a great option, the downside is Microsoft can have very steep licensing fees. The OS is not as stable as Unix based systems. This OS is not as optimized as others which increases resource needs during use. | Mobile devices are extremely limited in their usefulness as a server-side device for webhosting. While they can be good for small scale, low-traffic servers for applications such as a remote for a house built with IoT devices throughout. Beyond that, the hardware and memory are limited and almost impossible to scale making them unusable for high-traffic or resource intensive needs. |
| **Client Side** | For Mac client-side development, costs can be higher due to the need for Apple specific hardware for testing. There are Mac specific APIs that may require expertise. Mac will run all standard web browsers but also has its own proprietary browser – Safari – that should be tested. | Linux as client-side has some unique development considerations. Expertise in development and package management is needed. Testing time may increase due to multiple distributions of Linux being available that may require testing. Web browsers are typical and likely have no special testing needs. | Windows client-side development may have costs associated with license fees. Development time may take longer when testing due to multiple active Windows versions. Most development will be done with Windows technologies like .NET framework which may require additional expertise. | Considerations for mobile client-side development are increased due to the multitude of mobile devices. Different OS have their own development environments which may increase time to market as well as needing the expertise to work in each of these environments. As Android has multiple active versions, testing time may be a factor in overall time to release. |
| **Development Tools** | Development tools/IDE: Xcode (IDE for iOS/macOS), Visual Studio Code, JetBrains IDEs (e.g., IntelliJ IDEA for Java).  Programming languages:  Swift (iOS/macOS), Objective-C, Java, Kotlin (for Android). | Development tools/IDE: Visual Studio Code, JetBrains IDEs, Eclipse, Vim, Emacs.  Programming languages: Python, PHP, Ruby, Node.js, C/C++, and many others | Development tools/IDE: Visual Studio (for .NET), Visual Studio Code, JetBrains IDEs, Eclipse.  Programming languages: C/C++, .NET, JavaScript, Python, and others. | Development tools/IDE:  Xcode (iOS), Android Studio (Android), Visual Studio with Xamarin (cross-platform), React Native, Flutter.  Programming languages: Swift (iOS), Java/Kotlin (Android), Dart (Flutter), JavaScript/TypeScript (React Native). |

## 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**: A Linux based OS is the preferred operating platform for this project. This OS supports most of the popular IDE programs which will be familiar to the developers. As an open-source operating system, the Linux environment can be customized to suit the needs of “Draw It or Lose It”. Additionally, Linux has a strong and robust community that can help overcome technical challenges or be leveraged to improve aspects of the program. From a cost perspective, most Linux distributions are free.
2. **Operating Systems Architectures**: Linux, as a highly adaptable and versatile operating system, supports a range of architectures. For the client, it is important to know that Linux OS runs on the x86 architecture of modern computer hardware and the ARM architecture of mobile devices.
3. **Storage Management**: The most cost effective, reliable, and quickly expandable storage management will be a cloud hosted storage. This will allow The Gaming Room to only purchase what is required to optimally run the game, and, if in the future, modifications or popularity drives the need for more space, the client can purchase more storage without committing to the capital expenditure. The client should confirm that these cloud-based options offer redundancy to always ensure the availability of the data.
4. **Memory Management**: To best optimize the memory, a combination of techniques should be used. Caching the images prior to loading will reduce latency and improve the game experience. Then using garbage collection techniques to remove the old data, freeing up space for new data will enhance the performance of the system.
5. **Distributed Systems and Networks**: This game should be built on a client-server type platform. Having the system separated this way will allow for common communication protocols like HTTP and REST API to handle the various OS platforms that will be connecting for gameplay. This stateless design will maintain uniformity of the interface regardless of the client-side operating system. Having the system hosted on a cloud-based or serverless system will improve connectivity, reduce outages, and improve the player experience.
6. **Security**: Using the principle of least privilege will be the easiest way to help secure the system. Players will only need access to the program from the gameplay and have no need to gain access to any part of the system that could be used to modify the game or enter areas that could compromise the security of the client or the other players. This access control can be granted or restricted by role control in the RESTful API. Generally, the users can gain entry to the game via a username and password which should be encrypted.