

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 | 05/15/24 | Damian Alvarado | First iteration |
| 1.1 | 05/28/24 | Damian Alvarado | Added information into evaluation section |

## [Executive Summary](#_sbfa50wo7nsh)

Our client The Gaming Room would like to develop a web-based application named “Draw it or Lose it” for reference; inspiration is drawn from the 1980’s television game show “Win, Lose, or Draw”. where teams compete to decipher drawings, our solution involves utilizing a repository of stock drawings to present clues visually. This enables multiple teams to engage in four rounds of gameplay. By transitioning the game to a web-based platform, it becomes accessible across a multitude of devices, promising an enriched gaming experience. The outlined design aligns with the client's specifications, aiming to furnish a user-friendly and captivating gaming application.

## Requirements

* *A web-based platform must be utilized to promote compatibility between different devices and operating systems.*
* *Game should support more than one team with multiple players assigned to each team*
* *Game needs to have time limits where the drawings will be shown incrementally*
* *If a team fails their guess then the remaining teams need to have a chance to take a guess at solving the puzzle*

## [Design Constraints](#_2et92p0)

* The current configuration is only available for android and will need to be expanded to a web based distributed environment
* Singleton pattern will need to be used as only one instance of the game service can be active at a time

## [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)

Below is a description of the classes depicted in the UML class diagram for the Draw It or Lose It game application system:

The Entity class acts as the base superclass for all entities in the application, housing common attributes like 'id' and 'name' shared by its subclasses. Ensuring that every entity within the system is equipped with an identifier and a name.

Next the Game, Team, and Player classes extend the Entity superclass. These classes represent the core entities in the game application: a Game comprises multiple Teams, each consisting of multiple Players.

The GameService class is intricately connected with the Game class through a composition relationship, signifying its role in managing Game instances' lifecycle and holding references to multiple Game objects. Similarly, the Game class exhibits a composition relationship with the Team class, and the Team class, in turn, with the Player class.

Within the ProgramDriver class, serving as the main function's abode and the application's entry point, a singleton instance of GameService is instantiated. This denotes that only one instance of the GameService class will persist throughout the application's lifespan. The ProgramDriver class orchestrates the addition of games, teams, and players utilizing the GameService instance. Additionally, it relies on the SingletonTester class, as indicated by the <<uses>> arrow.

This UML class diagram aptly demonstrates fundamental object-oriented programming principles such as inheritance, encapsulation, and abstraction. The inheritance relationship between the Entity superclass and its subclasses (Game, Team, and Player) enables subclasses to inherit attributes and behaviors from the superclass, fostering code reusability and maintaining consistency. Encapsulation is evident in the GameService class, which conceals its attributes and exposes only methods for interacting with the data, ensuring data privacy and promoting abstraction. Abstraction is further emphasized by how the classes manage games, teams, and players, abstracting away unnecessary implementation details and focusing on essential interactions.

**"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** | Boasts a robust ecosystem and is favored by developers for its user-friendly nature. However, hardware costs are relatively higher, and scalability is somewhat limited compared to Linux and Windows alternatives. | The open-source operating system offers customization, flexibility, and a diverse array of software and tools. | Extensive software compatibility and a thriving developer ecosystem characterize the platform. It boasts broad hardware | Limited screen size and varied capabilities. Touchscreen based unit |
| **Client Side** | An intuitive and user-friendly interface helps mitigate the learning curve. However, developing and maintaining multiple clients can cost more. | While free for use and distribution, it's essential to factor in other expenses like hardware and tools. | The expenses to licensing may be higher than open-source alternatives. | Key considerations are responsive design and addressing connectivity limitations. Native functionalities such as camera, GPS, and push notifications should be incorporated. |
| **Development Tools** | Node.js and JavaScript are frequently utilized in software development. Popular IDEs such as VSCode and Xcode are commonly employed for coding. | IDEs such as VSCode, Atom, and Sublime Text. interfaces and package management systems like apt or yum. | C# and the .NET Visual Studio and JetBrains IDEs are commonly employed. | Java and JavaScript, programming languages such as Kotlin, Swift, and Objective-C are utilized. Development environments like Android Studio and Xcode, along with device emulators and simulators, are also employed. |

## 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:** To broaden the reach of Draw It or Lose It, transitioning to a web-based operating platform is recommended. This shift allows the game to be accessible on various devices with web browsers, ensuring consistency in the gaming experience across different platforms.

**2. Operating Systems Architectures:** The architecture for the chosen web-based platform revolves around client-server and web technologies. HTML, CSS, and JavaScript handle the client-side interface and interactions, while a multi-tier server-side architecture optimizes performance.

**3. Storage Management:** The recommended storage management system combines relational database management and cloud storage. RDBMS manages structured data, while cloud storage ensures scalability and accessibility of media assets.

**4. Memory Management:** Automatic memory management provided by modern web browsers is utilized, freeing developers from manual memory management tasks and ensuring efficient memory utilization.

**5. Distributed Systems and Networks:** Draw It or Lose It employs distributed software architecture and network connectivity to enable communication between platforms. A centralized server or cloud infrastructure facilitates game synchronization and real-time updates across devices.

**6. Security:** Stringent security measures are implemented to protect user information. Secure communication protocols, authentication, authorization mechanisms, and data encryption techniques safeguard sensitive data stored and transmitted across platforms.