

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 | 07/14/24 | Daniel Vidmar | Initial Creation |

**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 task is to develop a web-based game application, "Draw It or Lose It," similar to the 1980s television game "Win, Lose or Draw." The application will use pre-rendered stock images to provide clues for teams to guess the puzzle within a set time frame. The key solution involves setting up a robust environment that ensures unique identifiers for games, teams, and players while supporting multiple teams and players per game.

## Requirements

*The client requires that the game support one or more teams, each with multiple players. Unique names for games and teams must be enforced to avoid duplication. Additionally, only one instance of the game should be active in memory at any given time, necessitating unique identifiers for each game, team, and player. The game must be developed for a web-based environment to ensure accessibility and ease of use.*

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment imposes constraints such as ensuring data consistency and managing concurrent access. The use of web technologies introduces the need to support a vast array of browsers, which can impact the development and testing process. Unique identifiers must be used to prevent duplication and ensure that only one game instance exists at a time, requiring careful planning of the application's architecture to support scalability and real-time collaboration.

## [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 provided UML class diagram depicts the structure of the "Draw It or Lose It" game application within The Gaming Room project. The diagram includes classes such as ProgramDriver, SingletonTester, GameService, Game, Team, Player, and Entity, showcasing their attributes and methods along with their relationships.

* **ProgramDriver**: This class contains the main() method, indicating it is the entry point of the application. It uses the SingletonTester class to test the singleton instance of GameService.
* **SingletonTester**: The SingletonTester class has a method testSingleton(), which tests the singleton behavior of the GameService class.
* **GameService**: This is a singleton class responsible for managing game instances. It maintains lists of games and tracks the next IDs for games, players, and teams. The singleton pattern us used to ensure that only one instance of GameService exists, providing a centralized management point.
* **Game**: The Game class represents individual games, each containing a list of Team objects. It includes methods for adding teams and retrieving game information.
* **Team**: The Team class contains a list of Player objects, representing the players in each team. It provides methods to add players and retrieve team information.
* **Player**: The Player class represents individual players with attributes such as ID and name. It includes a method to retrieve player information.
* **Entity**: The Entity class serves as a base class for Game, Team, and Player, encapsulating common attributes such as ID and name, and methods for retrieving these attributes. This demonstrates inheritance, promoting code reuse and reducing redundancy.

**Object-Oriented Programming Principles**

* **Encapsulation**: Each class encapsulates its attributes and methods, providing a clear interface for interaction while hiding internal implementation details. For instance, GameService manages game-related data internally and exposes methods like addGame() and getGameCount() for interaction.
* **Inheritance**: The Entity class serves as a base class for Game, Team, and Player, demonstrating inheritance. This relationship allows these classes to share common attributes and methods, reducing code duplication and enhancing maintainability.
* **Singleton Pattern**: The GameService class follows the singleton design pattern, which ensures that there is only one instance of this class throughout the application. This pattern is crucial for centralized game management, providing a single point of access and control.
* **Relationships**: The diagram illustrates various types of relationships:
  + **Composition**: GameService has a composition relationship with Game, indicating that the GameService class is responsible for the lifecycle of Game objects.
  + **Aggregation**: Game aggregates Team objects, and Team aggregates Player objects, showing that games consist of teams, and teams consist of players.

These principles and relationships ensure the system is modular, scalable, and efficient in fulfilling the software requirements. The use of inheritance promotes reusability, the singleton pattern ensures centralized management, and encapsulation protects the integrity of the data.

**"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 operating systems provide a stable and user-friendly environment for hosting web-based applications, leveraging its robust Unix foundation. The advantages include seamless integration with development tools and a high level of security. However, its higher cost and less widespread use in server environments compared to Linux and Windows can be considered drawbacks. | | Linux is renowned for its stability, security, and flexibility, making it a preferred choice for hosting web-based applications. Its open-source nature allows for customization and cost-effectiveness. However, it may require more expertise to manage and configure, which could be a disadvantage for teams without strong Linux skills. | Windows servers are known for their ease of use, compatibility with a wide range of software, and comprehensive support. This makes them a popular choice for hosting web applications, especially in enterprise environments. However, licensing costs and security vulnerabilities are notable disadvantages compared to Linux. | Mobile devices do not typically serve as platforms for hosting web applications due to their limited resources and connectivity. However, they can interact seamlessly with web servers hosted on other platforms through mobile-optimized web applications. The primary challenge is ensuring compatibility and performance across various mobile operating systems and hardware configurations. |
| **Client Side** | Developing for Mac clients involves considering the higher cost of hardware and the expertise required for macOS-specific development. While Macs offer excellent performance and a smooth user experience, the market share is smaller compared to Windows, which may limit the audience. Development time can be longer due to the need for specific skills in Mac development. | | Supporting Linux clients involves considering the diversity of distributions and the varying levels of user expertise. Linux offers a cost-effective solution with high performance and security. However, the fragmented environment can lead to compatibility issues, and the user base is relatively smaller compared to other operating systems. | Developing for Windows clients involves leveraging a wide range of development tools and frameworks, with extensive support and documentation. Windows' large user base and compatibility with various software make it an attractive platform. However, development costs can be higher due to licensing fees and potential security issues. | Developing for mobile clients requires attention to the diversity of operating systems (iOS, Android) and device specifications. The mobile market is vast, offering significant reach, but it demands responsive design and optimization for different screen sizes and performance levels. Development time can increase due to the need for cross-platform compatibility and testing. |
| **Development Tools** | Macs support a wide range of development tools such as Xcode, which is essential for iOS and macOS development, and popular IDEs like IntelliJ IDEA and Visual Studio Code. Programming languages like Swift and Objective-C are tailored for Apple platforms. However, some tools may not be as well-supported as they are on Windows or Linux, potentially limiting development options. | | Linux supports a vast array of development tools, including IDEs like Eclipse and NetBeans, and programming languages such as Python, Java, and C++. The open-source ecosystem provides extensive resources and community support. However, some proprietary tools available on Windows or Mac may not have direct equivalents on Linux. | Windows supports a broad spectrum of development tools, including Visual Studio, which is highly regarded for its powerful features and support for multiple languages like C#, C++, and Python. The extensive availability of both open-source and proprietary tools provides flexibility. Nevertheless, the reliance on licensed software can increase costs. | Mobile development utilizes tools like Android Studio for Android and Xcode for iOS, along with cross-platform frameworks like Flutter and React Native. These tools support languages such as Java, Kotlin, Swift, and Dart. However, ensuring consistent performance across various devices and operating systems can be challenging and time-consuming. |

## 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**: I recommend using a Linux-based platform for expanding "Draw It or Lose It" due to its robustness, flexibility, and widespread support for web applications. Linux offers strong performance, scalability, and cost-effectiveness, making it ideal for diverse computing environments. Additionally, its open-source nature provides a high level of customization and community support.
2. **Operating Systems Architectures**: The Linux operating system architecture is based on a modular design with a monolithic kernel that efficiently manages system resources. It supports multitasking, multi-user capabilities, and a wide range of hardware. Linux's architecture ensures stability, security, and efficient performance, which are crucial for hosting web-based applications like "Draw It or Lose It."
3. **Storage Management**: For storage management, I recommend using a combination of SSDs for fast data access and retrieval, and a reliable file system like ext4 or XFS. These file systems provide excellent performance, data integrity, and scalability. Additionally, integrating a database management system such as MySQL or PostgreSQL will ensure efficient data storage and retrieval for the application's needs.
4. **Memory Management**: Linux employs sophisticated memory management techniques, including paging, swapping, and efficient use of virtual memory. The operating system dynamically allocates memory based on application needs, ensuring optimal performance and resource utilization. This will allow "Draw It or Lose It" to run smoothly, even with multiple concurrent users.
5. **Distributed Systems and Networks**: To enable communication between various platforms, the game can use RESTful APIs and WebSocket protocols, facilitating real-time interactions and data exchange. This approach ensures interoperability across different devices and operating systems. Robust network infrastructure with load balancing and redundancy will mitigate connectivity issues and enhance the reliability of the distributed system.
6. **Security**: To protect user information, implement end-to-end encryption for data transmission and use secure authentication mechanisms like OAuth. Linux offers robust security features, including SELinux for enforcing access controls and firewall configurations for network protection. Regular updates and security patches will help safeguard the system against vulnerabilities, ensuring user data remains secure across all platforms.