

<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 |
| --- | --- | --- | --- |
| 2.0 | <07/21/24> | <Jose Munoz> | |  | | --- | | Further update of the version software design document. |   Top of Form  Bottom of Form |

**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 aims to expand its popular Android game, "Draw It or Lose It," into a comprehensive web-based application accessible across multiple platforms. This expansion requires a scalable and maintainable architecture to handle unique naming conventions, single instance management, and support for multiple teams and players. Our design leverages proven software patterns such as Singleton and Iterator to ensure efficient and robust implementation. This document details the design constraints, UML domain model, and specific patterns employed to meet the client's requirements effectively.

## Requirements

**Multi-platform Support**: The game should run on various platforms.

**Unique Identifiers**: Ensure unique names for games, teams, and players.

**Singleton Instance**: Only one instance of the game should exist in memory.

**Scalability**: Handle multiple teams and players efficiently.

## [Design Constraints](#_2et92p0)

Developing "Draw It or Lose It" in a web-based distributed environment imposes several constraints:

**Performance**: The system must handle high concurrency, ensuring smooth gameplay without latency.

**Data Integrity**: Real-time data synchronization across platforms is crucial to maintain consistent game states.

**Security**: Robust security measures are necessary to protect user data and ensure secure communication.

**Scalability**: The architecture must support dynamic scaling to accommodate varying loads.

**Reliability**: The system must be highly available and fault-tolerant to provide a seamless gaming experience.

## [System Architecture View](#_ilbxbyevv6b6)

The system architecture for "Draw It or Lose It" is designed to be scalable, maintainable, and efficient, supporting a web-based distributed environment that can handle multiple platforms. The architecture includes several layers: the client layer with web and mobile clients, the application layer with web servers, game engine, and RESTful API, the data layer with a database server and caching system, and the infrastructure layer with a load balancer and content delivery network (CDN). Security measures such as authentication, encryption, and intrusion detection are implemented to protect user data. Additionally, CI/CD pipelines and monitoring tools are used for efficient deployment and continuous performance monitoring. This architecture ensures a seamless and secure gaming experience across various platforms.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for "Draw It or Lose It" consists of several classes, each representing a key component of the game application. The primary classes include ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player.

1. **ProgramDriver**:
   * Contains the main method to run the application.
2. **SingletonTester**:
   * Contains a method to test the Singleton pattern implementation.
3. **Entity**:
   * An abstract base class with common attributes id and name.
   * Methods include constructors, getters for id and name, and toString.
4. **GameService**:
   * Manages the collection of games.
   * Implements the Singleton pattern to ensure only one instance exists.
   * Attributes include lists of games and identifiers for games, teams, and players.
   * Methods include adding and retrieving games, teams, and players.
5. **Game**:
   * Inherits from Entity.
   * Manages a collection of teams.
   * Methods include adding teams and toString.
6. **Team**:
   * Inherits from Entity.
   * Manages a collection of players.
   * Methods include adding players and toString.
7. **Player**:
   * Inherits from Entity.
   * Represents individual players.
   * Methods include toString.

#### Relationships and Object-Oriented Principles

* **Inheritance**:
  + Game, Team, and Player classes inherit from the Entity class, demonstrating inheritance. This allows these classes to share common attributes (id and name) and methods, reducing redundancy and improving code maintainability.
* **Encapsulation**:
  + Each class encapsulates its attributes, providing public methods for accessing and modifying these attributes. This ensures that the internal state of objects is protected from unauthorized access and modification.
* **Singleton Pattern**:
  + GameService implements the Singleton pattern to ensure that only one instance of this class exists. This is crucial for managing the game state and ensuring consistent data across the application.
* **Iterator Pattern**:
  + Methods in GameService such as getGame, getTeam, and getPlayer use iteration to search through collections (games, teams, players) to ensure unique names and manage entities efficiently.

#### Efficiency in Fulfilling Software Requirements

* **Unique Identifiers and Names**:
  + By inheriting from Entity, all entities (games, teams, players) have unique identifiers and names, ensuring uniqueness across the application.
* **Single Instance Management**:
  + The Singleton pattern in GameService ensures that only one instance of the game management service exists, preventing conflicts and inconsistencies in game state management.
* **Scalability and Maintainability**:
  + The use of inheritance and encapsulation improves code scalability and maintainability. Changes to common attributes or methods can be made in the Entity class and will automatically propagate to Game, Team, and Player classes.
* **Efficient Collection Management**:
  + The iterator pattern allows for efficient management and retrieval of games, teams, and players, ensuring that operations like adding and finding entities are performed quickly and effectively.

This UML class diagram showcases a well-structured, object-oriented approach to designing the "Draw It or Lose It" application, ensuring that it meets the client's requirements efficiently and effectively.

**"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 OS is stable and efficient for development but can be more expensive. It offers a robust development environment but may have limited server-side application support compared to Linux. | Highly stable and cost-effective with extensive support for web servers. Ideal for hosting due to its scalability, security, and wide range of development tools. Preferred for server-side applications. | Widely used but can be more prone to security issues and less stable for server applications. Good for development but may incur higher licensing costs and maintenance. | Not suitable for hosting server applications due to limited resources and potential instability. Better suited for client-side applications. |
| **Client Side** | High development cost but great for graphics and multimedia applications. Development on Mac ensures high-quality output but may require more time and expertise. | Cost-effective and flexible, but requires expertise. It supports a variety of programming languages and tools but may have a steeper learning curve. | Familiar environment, good for development but licensing costs can be high. Widely used, providing broad compatibility and a wide range of development tools. | Essential for reaching mobile users but requires responsive design and mobile-specific features. Development requires consideration of various screen sizes and capabilities. |
| **Development Tools** | Xcode, IntelliJ IDEA, Java, and other development tools. Mac is excellent for iOS development and general software development with its rich ecosystem. | Eclipse, IntelliJ IDEA, Java, and various open-source tools. Ideal for web server development due to its flexibility and powerful command-line tools. | Visual Studio, IntelliJ IDEA, Java. Well-suited for desktop and web applications with robust tools like Visual Studio. | Android Studio for Android, Xcode for iOS. Essential for mobile app development, providing emulators and tools specific to mobile platforms. |

## 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**: Linux is recommended for server-side deployment due to its stability, cost-effectiveness, and extensive support for web servers. It ensures a robust and scalable environment for hosting the game.
2. **Operating Systems Architectures**: Linux architecture is preferred for its flexibility, performance, and security. It supports efficient resource management and is highly compatible with various web technologies.
3. **Storage Management**: MySQL or PostgreSQL are recommended for reliable data management. These databases offer robust performance, scalability, and security features suitable for managing game data.
4. **Memory Management**: Linux uses advanced memory management techniques such as virtual memory and efficient caching to optimize performance. This ensures smooth and responsive gameplay by effectively managing system resources.
5. **Distributed Systems and Networks** Implement redundancy and failover mechanisms to ensure high availability and reliability. Use a load balancer to distribute traffic efficiently across servers.
6. **Security**: Implement SSL/TLS for data transmission to ensure secure communication. Use secure authentication methods like OAuth and regularly update and patch systems. Employ encryption for sensitive data storage and ensure robust access control mechanisms to protect user information.