

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)**-4**

[**Domain Model**](#_2s8eyo1) **4-5**

[**Evaluation**](#_17dp8vu) **6**

[**Recommendations**](#_26in1rg) **7-8**

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/16/2024 | Raul Ochoa | First document version. |
| 1.1 | 12/01/2024 | Raul Ochoa | Updated Evaluation Table |
| 1.2 | 12/15/2024 | Raul Ochoa | Updated Recommendations Section |

## [Executive Summary](#_35nkun2)

Creative Technology Solutions (CTS) has been tasked with developing a web-based, multi-platform version of "Draw It or Lose It" for The Gaming Room. The existing game is currently only available as an Android application. The Gaming Room wants to expand its reach by creating a version that works across web browsers, mobile devices, and other platforms.

The goal of this project is to leverage Agile methodologies and design patterns to create a scalable, efficient, and user-friendly game application. The key requirements include supporting multiple teams, allowing each team to have multiple players, and ensuring that names for games, teams, and players are unique. Additionally, only one instance of the game should exist in memory at any time to optimize resources.

The solution involves implementing a robust back-end system using Java with design patterns such as Singleton and Iterator. The Singleton pattern ensures that only one instance of *GameService* exists, while the Iterator pattern helps efficiently manage lists of games, teams, and players. By using these patterns, the game can handle real-time data processing and scalability requirements, ensuring a smooth user experience across various platforms.

## Requirements

The client has specified the following business and technical requirements for the project:

* The game application must support one or more teams, with each team having multiple players.
* Game, team, and player names must be unique to avoid conflicts.
* Only one instance of the game application should exist at a time, ensuring that memory resources are efficiently utilized.
* The game must be web-based and compatible across multiple platforms, including desktop and mobile devices.
* The application must be scalable to support future growth and updates.

## [Design Constraints](#_1ksv4uv)

Developing a web-based, distributed game application like "Draw It or Lose It" presents several design constraints that must be considered:

1. **Scalability**: The game must be able to handle multiple users simultaneously across different platforms (e.g., web browsers, iOS, Android). This requires efficient resource management to ensure smooth gameplay without performance degradation.
2. **Concurrency**: Since the game involves real-time participation from multiple teams and players, concurrency management is essential. Implementing a Singleton pattern for *GameService* ensures that only one instance manages the game state, reducing the risk of conflicts.
3. **Platform Compatibility**: The game must function consistently on various platforms, including Linux, Windows, Mac, and mobile devices. Each platform has its own set of libraries, system calls, and performance characteristics that must be taken into account during development.
4. **Security**: Protecting user data is crucial, especially since the game will be accessible over the internet. This includes securing communication between clients and servers, as well as preventing unauthorized access to game data.
5. **Data Integrity**: Ensuring that names (games, teams, players) are unique requires efficient searching and validation mechanisms. The use of the Iterator pattern helps prevent duplicates by iterating through existing entries before creating new ones.
6. **Resource Management**: The game must be optimized for memory usage, especially on mobile devices with limited resources. Implementing best practices for memory management ensures that the game remains responsive, even on devices with lower specifications.

## [Domain Model](#_2jxsxqh)

**Class Structure**:

* The **Entity** class serves as a base class for other entities in the game. It defines the common attributes **id** and **name**, which are inherited by the **Game**, **Team**, and **Player** classes. This approach promotes code reusability and reduces duplication.
* The **GameService** class is responsible for managing the entire game environment. It implements the **Singleton pattern** to ensure that only one instance of **GameService** exists in memory at any given time. This prevents conflicts when multiple users access the game concurrently.

**Object-Oriented Principles**:

* **Encapsulation**: Each class encapsulates its own data (such as **id** and **name**) and provides access through getter methods. This ensures that data is protected and can only be modified in controlled ways.
* **Inheritance**: The **Game**, **Team**, and **Player** classes inherit from the **Entity** class, sharing common attributes and behaviors. This reduces redundancy and simplifies maintenance.
* **Polymorphism**: By leveraging the **toString**() method in each class, the application can easily convert objects into string representations, allowing for flexibility in displaying information.

**Design Patterns**:

* The **Singleton pattern** is used in **GameService** to maintain a single instance of the game service throughout the application's lifecycle. This is crucial for managing the game's state and ensuring efficient memory usage.
* The **Iterator pattern** is used to search through lists of games, teams, and players. This pattern helps efficiently find existing entities by iterating over collections, ensuring that names remain unique.

**"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](#_z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Stable and reliable but less commonly used for hosting due to higher costs and fewer hosting options. | Highly popular for server hosting; open-source, cost-effective, and supports servers like Apache and Nginx. | Widely used in corporate settings; compatible with .NET but less resource-efficient than Linux for web apps. | Requires lightweight, optimized servers; cloud-based servers cater to mobile needs (iOS and Android). |
| **Client Side** | Familiarity with Xcode and macOS increases development costs. | Cost-effective with diverse tools; requires higher technical expertise, increasing time and costs. | Familiar to many developers with tools like Visual Studio; generally more expensive than Linux. | Cross-platform tools (React Native, Flutter) reduce costs and time for iOS and Android compatibility. |
| **Development Tools** | Supports IDEs like Eclipse and IntelliJ; requires configuration for web-based projects. | Compatible with versatile tools like Eclipse and IntelliJ; powerful command-line utilities for developers. | Tools like Visual Studio simplify setup for Java apps but have higher licensing costs. | Android Studio and Xcode essential; cross-platform frameworks streamline iOS and Android development. |

## Recommendations

#### 1. Operating Platform

**Recommendation:** Linux is the recommended operating platform for server-side deployment of "Draw It or Lose It."

**Justification:**

* **Scalability:** Linux supports high-performance web servers like Apache and Nginx, making it perfect for handling multiple simultaneous game sessions.
* **Cost-Effectiveness:** As an open-source platform, Linux eliminates licensing costs, reducing the total cost of ownership.
* **Compatibility:** Linux integrates well with Java-based servers, allowing execution of the game's backend.
* **Community Support:** Large community resources and active development ensure reliable updates and troubleshooting.

#### 2. Operating Systems Architectures

The Linux operating platform has a modular architecture, which provides:

* **Flexibility:** Developers can load only the necessary modules, optimizing resource usage.
* **Efficiency:** The monolithic kernel design ensures faster execution by minimizing communication overhead between kernel and user processes.
* **Robust Concurrency Management:** Features like POSIX threads and process isolation help with smooth operation of concurrent game sessions.

#### 3. Storage Management

**Recommendation:** AWS S3 is an ideal storage management system for "Draw It or Lose It."

**Justification:**

* **Scalability:** AWS S3 supports dynamic scaling, ensuring storage needs grow with the user base.
* **Data Integrity:** Built-in redundancy mechanisms help prevent data loss.
* **Access Control:** Supports access policies to control game data visibility.
* **Performance:** Provides low-latency access to game assets, enhancing the user experience.

#### 4. Memory Management

Linux uses advanced memory management techniques, including:

* **Virtual Memory:** Optimizes physical memory usage by allocating additional memory from disk when needed.
* **Garbage Collection:** Java’s built-in garbage collector efficiently manages memory allocation and deallocation, preventing memory leaks during gameplay.
* **Caching Mechanisms:** Linux’s page cache ensures frequently accessed data is stored in memory, improving the game's responsiveness.

#### 5. Distributed Systems and Networks

To ensure seamless communication between platforms:

* **Implementation:** Use RESTful APIs to facilitate communication between the game server and client devices (web, Android, iOS).
* **Dependencies:**
  + **Load Balancing:** Use load balancers to distribute traffic evenly, preventing server overloads.
  + **Failover Mechanisms:** Use automatic failover systems to ensure uninterrupted service during outages.
* **Connectivity:** Use cloud-based services like AWS or Azure to provide stable and scalable networking solutions.

#### 6. Security

To protect user information:

* **Data in Transit:** Use HTTPS and TLS to encrypt communication between clients and the server.
* **Data at Rest:** Store sensitive game and user data in encrypted databases like AWS RDS with encryption.
* **Authentication:** Use OAuth or Basic Authentication for secure user login.
* **Role-Based Access Control (RBAC):** Restrict game data access to authorized roles, ensuring that only permitted users can view or modify critical information.
* **Regular Audits:** Conduct routine security audits and updates to patch vulnerabilities promptly.

By implementing these recommendations, "Draw It or Lose It" can efficiently expand to multiple platforms while maintaining security, scalability, and a seamless user experience.