

Draw It or Lose It – Web Game Expansion

# **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 | 05/25/2025 | Austin Bryan | Initial software design document |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has tasked Creative Technology Solutions with expanding their existing Android only game, Draw It or Lose It, into a web-based application accessible across multiple platforms. This game, inspired by the class Win, Lose or Draw, requires careful coordination of game flow, rendering of drawing-based clues, and real-time user interactions. The proposed solution involves building a web-based distributed version that maintains the game logic and architecture while introducing scalable and platform agnostic delivery. The project will employ object-oriented design, singleton and iterator design patterns, and web friendly development tools to ensure performance, scalability, and maintainability.

## Requirements

**Business Requirements**

* Expand Draw It or Lose It to a web-based environment
* Support multiple teams and multiple players per team
* Ensure name uniqueness for games and teams
* Provide cross platform access and user experience

**Technical Requirements**

* Only one instance of the game should exist in memory at any time (singleton)
* Search and verification of unique names using collection traversal (iterator)
* Common object structure for Game, Team, and Player all inheriting from the Entity class
* Compatible with modern web hosting and client environments

## [Design Constraints](#_2et92p0)

1. **Singleton Enforcement:** Only one game instance can exist in memory. This requires implementing the singleton design pattern in the GameService class. It limits the flexibility of the game instance creation but ensures global consistency and shared state management
2. **Name Uniqueness via Iterators:** Team and game names must be unique, requiring validation through existing collections. This constraint demands the user of the iterator pattern to loop through and compare existing game or team instances.
3. **Web-Based Deployment:** The software must run in a distributed, platform-agnostic environment. This eliminates native Android tools and libraries, and requires the use of web compatible languages, frameworks, and stateless architecture.
4. **Inheritance via Entity Class:** The application’s design requires the creation of an Entity base class to store shared attributes. This constraint enforces inheritance and promotes DRY principles, but also requires careful handling of casting and polymorphism in collections

## [System Architecture View](#_ilbxbyevv6b6)

Not required for this project.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram shows a base class called Entity, which defines the shared attributes id and name. Three primary classes inherit from this base: Game, Team and Player. The Game class can contain multiple Team instances, and each Team contains multiple Player instances. The GameService class manages the creation and retrieval of these instances and enforces unique names and singleton behavior.  
  
**OOP Principles Demonstrated:**

* **Inheritance:** Game, Team, and Player all inherit from Entity, reducing redundancy
* **Encapsulation:** Each class hides its internal data while exposing relevant methods
* **Abstraction:** Shared behaviors are grouped in the Entity class to promote reusability
* **Polymorphism:** Entities can be referenced through their base class when traversing collections

This structure allows for scalable maintainable code that matches the business need for flexibility, while enforcing consistent behavior across all components.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS supports server deployment using Apache, Tomcat, or Node.js. However, it's not commonly used for production servers because it requires Apple hardware, which is expensive. The operating system itself doesn’t have extra licensing fees, but the hardware cost makes it less ideal for large-scale hosting. | Linux is the most popular platform for server hosting. It supports deployment using Apache, NGINX, Tomcat, and other frameworks. It is free, open-source, secure, and highly scalable, making it ideal for web-based applications that need to support thousands of users. There are no licensing costs, which makes it cost-effective for The Gaming Room. | Windows Server supports web-based deployment using IIS, Apache, or Tomcat. It offers strong scalability and works well with .NET-based applications. Its user-friendly GUI makes server management easier. However, Windows Server requires a paid license, which would increase costs for The Gaming Room compared to Linux. | Mobile platforms like iOS and Android are not designed for server hosting and do not support full server deployment. They can run limited local services for testing, but not at a scale suitable for production. The backend for the application should be hosted on a remote server. There are no server licensing costs for mobile operating systems because they are not used for backend hosting. |
| **Client Side** | macOS clients will access the application through Safari or Chrome, so a responsive HTML5 design is required to ensure compatibility and proper scaling. Developers must test layout behavior on Retina displays to avoid UI issues. No extra licensing is needed, but testing on macOS hardware is required, which can add to cost and setup time. Moderate front-end development expertise is needed for cross-platform browser support. | Linux supports modern browsers like Chrome and Firefox, making it fully compatible with responsive HTML5 applications. Development is low-cost due to open-source tools, but developers may need command-line skills for setup and testing. Since there are many Linux distributions, testing should be done across common ones to ensure consistent behavior and compatibility. | Windows supports all major browsers like Edge, Chrome, and Firefox, making it easy to build and test a responsive HTML5 interface. It offers the highest accessibility for most development teams due to its widespread use and available hardware. Testing is straightforward on Windows machines, and developers typically require less training to work with this platform. | Mobile support requires a responsive HTML5 UI that adapts to different screen sizes and hardware. Developers must test layouts and performance across both Android and iOS browsers. Cross-platform tools like responsive frameworks can help speed up development, but moderate to high expertise is needed to handle mobile-specific quirks and ensure a smooth user experience. |
| **Development Tools** | Development for macOS and iOS often uses Xcode for native apps and Safari for browser testing. Cross-platform web apps can be developed using JavaScript/HTML5 in editors like Visual Studio Code or IntelliJ. Most tools are free, but macOS hardware is required, which may increase cost. Some Apple-specific knowledge may be needed, and teams without access to Mac hardware could face delays. | Linux supports a wide range of free tools like Eclipse, NetBeans, IntelliJ (Community), and frameworks like Apache and Node.js. It’s well-suited for Java-based web development and backend services. There are no licensing costs. Most developers familiar with cross-platform tools can adapt easily, but Windows- or Mac-only developers may need some retraining. | Windows supports tools like Visual Studio, IntelliJ, Eclipse, and languages like C#, JavaScript, and Java. It has strong support for .NET-based development and web applications. Many tools have free versions, but some features in Visual Studio may require a paid license. Most dev teams are already familiar with Windows, so little training is needed, and testing is easy on widely available Windows hardware. | Mobile development uses tools like Android Studio (Java/Kotlin), Xcode (Swift), and cross-platform frameworks like React Native, Flutter, or Unity. Separate teams may be needed for native Android and iOS apps unless hybrid tools are used. Most tools are free, but publishing to app stores may involve fees, especially for Apple. |

## Recommendations

1. **Operating Platform:** Linux is the recommended operating platform for server-side development. It offers unmatched flexibility, free to use with no licensing costs, and robust compatibility with Java based backend tools like Tomcat and Spring Boot. Linux is also highly scalable, secure and supported by major cloud providers like AWS and Azure, making it ideal for handling increasing player traffic without adding overhead.
2. **Operating System Architectures:** Linux uses a modular, monolithic kernel architecture which efficiently handles I/O operations, process scheduling, and memory management. For *Draw It or Lose It*, a three tier architecture (presentation, logic, data) can be used. The web front-end runs in the browser, the logic tier handles the game state and rule enforcement via REST APIs while the last tier, the data tier, manages persistent storage in MySQL or PostgreSQL.
3. **Storage Management:** MySQL is recommended for structured storage of player profiles, scores and team data. For image assets, use a cloud based object store such as AWS S3 or Azure Blob storage, along with access control. Additionally, use chunk based deduplication techniques (Xia et al., 2016) to reduce asset redundancy. These strategies enable scalable, maintainable storage that aligns with both web and mobile platform constraints.
4. **Memory Management:** Java’s JVM (Java Virtual Machine) handles memory through automatic garbage collection and memory pools. Using the Singleton pattern reduces memory duplication by ensuring only one game controller instance exists. Techniques like LRU (Least Recently Used) caching and image subsampling (Android Developers, 2024) ensure memory usage is optimized, especially during rendering of images mid-game.
5. **Distributed Systems and Networks:** A stateless RESTful API architecture will ensures seamless operation across distributed systems. Load balancers and replication strategies improve reliability and reduce latency. Game sessions and player state can be managed with token based authentication, JWT (JSON Web Tokens) and session stores like Redis, ensuring continuity even cross multiple devices or network outages.
6. **Security:** End-to-end encryption via HTTPS ensures secure data transition. Password hashers such as bcrypt (Blowfish-based Crypt) or Argon2 can be used for extra security. Role based controls to restrict game state changes to authorized users should also be used. Input validation and rate limiting prevent abuse, and JWT ensures secure sessions continuity. Linux’s built-in firewall tools (like iptables or UFW (Uncomplicated Firewall) and cloud level IAM (Identity and Access Management) policies provide layered protection

References

Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., Zhang, Y., & Liu, Q. (2016). *FastCDC: A fast and efficient content-defined chunking approach for data deduplication*. In *Proceedings of the 2016 USENIX Annual Technical Conference* (pp. 101–114). <https://www.usenix.org/system/files/conference/atc16/atc16-paper-xia.pdf>

Android Developers. (2024, January 3). *Loading large bitmaps efficiently*. <https://developer.android.com/topic/performance/graphics/load-bitmap>