

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 07/17/2024 | Timothy Johnson | Initial draft created |
| 1.1 | 8/2/2024 | Timothy Johnson | Updated Evaluation |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) has undertaken the task of developing a web-based version of The Gaming Room's Android game, Draw It or Lose It. The game requires functionality for multiple teams and players, ensuring each game, team, and player instance remains unique. To achieve this, CTS has implemented the singleton pattern in the Game Service class for managing game instances and utilized the iterator pattern to enforce uniqueness in team and player names. These design choices ensure efficient management and prevention of duplicate entities within the game environment.

## Requirements

Business Requirements:

* Develop a web-based version of the existing Android game "Draw It or Lose It."
* Support multiple teams with multiple players per team.
* Ensure each game instance, team, or player occurs only once.
* Maintain compatibility with the existing Android deployment of the game.
* Review and possibly extend existing APIs for mobile usage.

Technical Requirements:

* Use Java as the primary programming language for web deployment, leveraging its compatibility with the Android SDK.
* Implement design patterns such as Singleton for Game Service to ensure a single instance, and Iterator to prevent duplicates in game, team, and player creation.
* Use UML for designing the system, including classes like Game, Team, Player, and Entity with inheritance and polymorphism.
* Ensure encapsulation and abstraction in object creation processes.
* Consider scalability and maintainability in the design to accommodate future expansions.

## [Design Constraints](#_2et92p0)

The primary constraint for CTS is transitioning Draw It or Lose It from its current Android platform to a web-based environment. Java has been selected as the development language due to its compatibility with both Android and web applications, easing the integration effort and leveraging existing Android APIs for mobile support. This choice ensures continuity in development and maintenance processes while extending the game's reach to web users.

## [System Architecture View](#_ilbxbyevv6b6)

* Client Interface
* Server-Side Components
* Web Server
* Application Layer
* Integration with Existing Android Deployment
* Communication
* Storage
* Security Considerations
* Scalability and Performance

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram illustrates a robust domain model for Draw It or Lose It. The Game, Team, and Player classes inherit from the Entity class, which provides common attributes such as id and name. This design exhibits object-oriented principles like inheritance and encapsulation, facilitating structured entity management. The use of iterator patterns in Game Service ensures unique entity creation, demonstrating effective design to meet the software requirements.

**"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** | Server Based Deployment: macOS Server or macOS with third-party server software. Advantages: Seamless integration with Apple ecosystem. Good for smaller scale or development/testing environments. Weakness: Limited scalability and performance for high-traffic applications. Higher costs for scaling up. macOS Server: $19.99 (one-time purchase). Additional costs for high-end hardware and scaling. | Server Based Deployment: Linux server distributions (ex. Ubuntu, CentOS)  Advantages: Open source and cost effective. Highly flexible and customizable. Strong security and performance.  Weakness:  Requires expertise for configuration and maintenance. Compatibility issues with some proprietary software.  Free (open source)  Costs may include support and enterprise features from third-party vendors. | Server Based Deployment: Windows Server editions (ex. Windows Server 2022)  Advantages:  Integration with Microsoft tools and services. Strong enterprise support. Good performance and scalability.  Weakness: Higher licensing costs. Requires regular updates and maintenance.  Windows Server: Approximately $800–$6,000+ depending on edition and licensing model. | Server Based Deployment:  Web apps accessed via browsers or native apps with web views.  Advantages: Portable and accessible from anywhere. Utilizes device features like cameras and GPS.  Weakness: Limited screen size and processing power. Variability in connectivity and user experience.  No direct licensing costs for web access. App store fees for native apps: $99/year (iOS) and a one-time fee of $25 (Google Play). |
| **Client Side** | Ensure compatibility with major browsers on macOS (ex. Safari, Chrome, Firefox).  Testing should cover different browser versions and resolutions. Consider specific macOS features. | Ensure compatibility across different Linux distributions and web browsers (ex., Firefox, Chrome). Requires thorough testing across various Linux distributions. Utilize responsive design principles. | Ensure compatibility with major browsers on Windows (ex. Edge, Chrome, Firefox). Testing across different browser versions and screen sizes is essential. Integration with Windows-specific features if needed. | Ensure compatibility with mobile browsers and native app environments (iOS and Android). Responsive design and touch interactions are crucial. Consider app store submission requirements and guidelines. |
| **Development Tools** | Xcode (iOS/macOS), IntelliJ IDEA, Eclipse, Visual Studio Code. Languages: Swift, Objective-C, Java, Kotlin.  Impact: Requires familiarity with Apple development tools and languages. Potential need for separate team or additional training for Apple-specific development. | IntelliJ IDEA, Eclipse, Visual Studio Code, GCC, Make. Languages: Java, Python, PHP, Ruby, Node.js.  Impact:  Expertise required in Linux environment and open-source tools. Likely lower costs but may require more diverse skill sets. | Visual Studio (.NET), IntelliJ IDEA, Eclipse, Visual Studio Code. Languages: C#, Java, Python, PHP, JavaScript.  Impact:  Requires knowledge of Microsoft development tools and environments. Higher licensing costs but strong support for enterprise environments. | Android Studio (Android), Xcode (iOS). Languages: Java, Kotlin (Android), Swift, Objective-C (iOS).  Impact:  Need for expertise in mobile development for both iOS and Android. Increased complexity and costs due to multiple platforms and app store submissions. |

## 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 because it's strong, cost-effective, and has good community support, ensuring Draw It or Lose It can grow easily.
2. **Operating Systems Architectures**: Linux's open-source nature provides flexibility and security, making it easy to scale and manage Draw It or Lose It across different computers.
3. **Storage Management**: Using MySQL or PostgreSQL on Linux ensures reliable data management and scalability, important for handling more players and data in Draw It or Lose It.
4. **Memory Management**: Linux manages memory well with optimizations, ensuring Draw It or Lose It runs smoothly and uses resources effectively.
5. **Distributed Systems and Networks**: By using RESTful APIs and message queues, platforms in Draw It or Lose It can communicate smoothly in real-time, even if there are network issues.
6. **Security**: Linux has strong security with features like SELinux and iptables, keeping player information safe across all parts of Draw It or Lose It.