

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 |
| --- | --- | --- | --- |
| 3.0 | 08/18/2024 | Aubry Moody | Updated recommendations |

**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 objective is to outline the development strategy for transitioning the current Android based game, Darw It or Lose It, to a web-based platform. The primary goal is to turn this game into a multi-platform game experience while ensuring unique identification for each game instance, team, and player. The solution involves using design patterns like Singleton for a single instance of the game and Iterator for navigating through teams and players.

## Requirements

The game application must support multiple teams, with each team having several players. It is essential that all teams and players have unique identifiers to avoid any conflicts and ensure accurate tracking, and the application must do this by using unique identifiers for each game instance, team, and player. The application also must maintain a single instance of the game in memory and for this the Singleton design will be used. Lastly the Iterator pattern will be implemented to navigate through collections of teams and players efficiently.

## [Design Constraints](#_2et92p0)

Developing the game in a web-based environment we must ensure compatibility across multiple platforms and devices, which affects the user interface and backend systems. The application must be designed to handle many different web browsers and operating systems, which introduces challenges in maintain consistent functionality. Connectivity issues must also be considered and making robust measures to protect user data and prevent unauthorized access. Additionally the application needs to be able to accommodate high user loads and maintain performance under varying conditions.

## [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)

* **Entity:** A base class representing common attributes shared by all game elements
* **Game:** Extends ‘Entity’ and contains a list of ‘Team’ objects. Manages teams and ensures game specific functionalities
* **Team:** Extend ‘Entity’ and contains a list of the ‘Player’ objects. Manages players and team specific functionalities.
* **Player:** Extends ‘Entity’ and represents the individual players.

‘Game’, ‘Team’ and ‘Player’ inherit from the ‘Entity’ base class, enabling code reuse and simplifying management of common attributes. Each class encapsulates its specific attributes and methods. The Iterator pattern is used in the ‘Game’ and ‘Team’ classes to navigate through list of ‘Team’ and ‘Player’ object efficiently.

**"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** | Are less common for web hosting. Potential for high cost and less support for server-side technologies. However, they offer a stable environment with good performance and integrates easily with the Apple system. | Widely used for web servers due to its stability, performance, and cost effectiveness. | Provides support and integration with enterprise tools but can be expensive. They are also used widely in business environments. | Not suitable for web servers due to limited resources and the need for consistent uptime and reliability. |
| **Client Side** | Requires additional setup for cross platform compatibility. MacOS users can run web applications through browsers like Safari, Chrome, and Firefox. Development for iOS will need tools like Xcode. | Supports a variety of development tools and languages, providing a cost-effective environment for development. Web applications can be run on Chrome. | Has a wide range of development tools and extensive support. Web applications can be run on browsers like Chrome, Edge, and Firefox. Development for Windows will need tools like Visual Studio. | Requires separate tools for Android and iOS. Development for Android uses tools like Android studio and Java while iOS uses Xcode and Swift. Responsive web design will be needed to ensure compatibility between all mobile devices. |
| **Development Tools** | Xcode is the primary IDE. It supports Swift, Objective-C, and C++. Mac development tools provide good support for developing macOS and iOS applications. | Popular IDEs include Visual Studio Code, Eclipse, IntelliJ IDEA. It supports a wide range of programming languages such as, Java, Python, and Ruby. Open-source tools are cost effective. | Visual Studio and Eclipse are the primary IDEs. It supports many languages such as, C#, .NET, and Java. | For Android, Android Studio is common. For iOS, Xcode and Swift/Objective-C are used. |

## 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 a great choice for its stability, scalability, and security. Linux supports a wide range of web technologies and is customizable. Also, Linux distributions offer long term support.
2. **Operating Systems Architectures**: Linux is known for its flexible architecture, which is well suited for web-based applications. It supports a wide range of file systems and has built-in security features like user permissions. Linux core functions run in a single address space providing efficient performance.
3. **Storage Management**: Use MySQL for relational database management, providing efficient storage, retrieval, and management of game data. MySQL is perfect for the “Draw It or Lose It” application because it supports large databases, which is essential as the games user base expands, and its replication features can be used for load balancing ensuring high availability.
4. **Memory Management**: Linux uses efficient memory management techniques like virtual memory and paging, ensuring that the application performs optimally, under various load conditions. Because Linux utilizes virtual memory it can allow applications to use more memory than physically available by swapping unused memory to disk. Paging helps manage the memory more efficiently by dividing memory into blocks and only loading necessary pages. Since the “Draw It or Lose It” application will be used for many users across many different platforms it will also help that Linux allows overcommitting memory, which can be beneficial for handling large workloads.
5. **Distributed Systems and Networks**: Implementing load balancing can distribute traffic across multiple servers, ensuring that no server becomes bottlenecked. Breaking the game’s functionality into microservices that can run independently could also allow for easier scaling and maintenance. Keeping redundancy in mind during the design process will help ensure that if a component fails the others can continue to operate that way downtime will be at a minimum.
6. **Security**: Use robust authentication and authorization mechanisms to protect user data and ensure secure access to game functions. Also, regularly update and patch server software to mitigate vulnerabilities. Consider using OSSEC to monitor and respond to potential security threats.