

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 | 10/15/2023 | Elijah Paulk | 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)

This document outlines the software design for the Draw It or Lose It. The application aims to address the client's requirements by implementing a game system with unique names for games, teams, and players. It utilizes object-oriented principles, the singleton pattern for the **GameService** class, and the iterator pattern to ensure uniqueness of names. The design constraints include the development of a web-based distributed environment. The following sections detail the architecture and evaluation of the development platforms.

## Requirements

The client's requirements include implementing a game system with unique names for games, teams, and players. The application should use object-oriented principles, the singleton pattern, and the iterator pattern to fulfill these requirements.

## [Design Constraints](#_2et92p0)

The design constraints involve developing the game application in a web-based distributed environment. This constraint implies that the application should be accessible over the internet and handle interactions from various devices and platforms. The architecture must support these requirements efficiently.

## [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)

The provided UML class diagram depicts the structural organization of the software and the relationships between its classes.

At its core, the **Entity** class functions as the foundation for other classes, encompassing common attributes such as **id** and **name** and adhering to key object-oriented principles like encapsulation. **GameService** stands out as a Singleton pattern class, responsible for overseeing games, teams, and players. This class showcases the Singleton design pattern, ensuring that only one instance of it exists, while also implementing encapsulation for controlled data access.

**Game** and **Team** classes inherit from **Entity**, showcasing the concept of inheritance, and they effectively manage their respective elements, teams, and players.

The **Player** class extends **Entity** and focuses on initializing player-specific attributes. Overall, this UML diagram succinctly communicates class relationships, inheritance, and aggregation, facilitating an efficient software design.

**"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 is known for its stability and security, making it suitable for hosting web-based software applications. Weaknesses could include limited scalability for high-traffic applications. | Linux excels in server configurations and can be tailored to specific application requirements. However, it may require more expertise to set up and maintain compared to Mac or Windows. | Windows Server is well-suited for hosting web applications, especially those built on Microsoft technologies. Advantages include ease of use and integration with Microsoft tools and technologies. However, licensing costs can be a consideration, and it may not be as cost-effective as Linux. | Mobile devices are not typically used for hosting web-based software applications. They are primarily client devices, and while they can access web applications, they are not suitable for serving applications to multiple users simultaneously. |
| **Client Side** | Supporting Mac clients may require considering the cost of Mac hardware and the expertise needed to develop and test on Mac OS. Mac users often expect a seamless user experience, and applications should be optimized for Mac's user interface guidelines. | Supporting Linux clients can be cost-effective as Linux is open-source. However, it may require additional time and expertise to ensure compatibility with various Linux distributions. Linux users appreciate software that is well-documented and follows open standards. | Windows clients are prevalent in the desktop computing world. Developing for Windows may require consideration of licensing costs and testing across different Windows versions. Expertise in Windows development tools like Visual Studio is essential. | Supporting mobile devices, such as smartphones and tablets, is crucial for reaching a broad user base. Development for mobile devices involves considerations of various platforms (iOS, Android) and screen sizes. Costs can vary depending on the choice of development tools, such as Android Studio. |
| **Development Tools** | Mac development often involves using Xcode, Apple's integrated development environment (IDE). Objective-C and Swift are the primary programming languages for Mac and iOS development. | Linux development can be done using a variety of programming languages and tools, including C/C++, Python, Java, and various integrated development environments (IDEs) like Visual Studio Code. | Windows development often involves using Visual Studio, which supports languages like C#, C++, and .NET. Windows also supports development in languages like Python and Java. | Developing for mobile devices typically requires platform-specific tools and languages. For iOS development, Xcode and Swift/Objective-C are used, while Android development involves Android Studio and Java/Kotlin. |

## 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**: I recommend using a web-based platform that allows easy expansion to other computing environments. This ensures compatibility with various devices and platforms.
2. **Operating Systems Architectures**: The chosen platform should support modern operating system architectures, ensuring scalability and performance.
3. **Storage Management**: Implement a robust storage management system suitable for the recommended operating platform to efficiently handle game data.
4. **Memory Management**: Utilize memory management techniques supported by the chosen platform to optimize resource usage.
5. **Distributed Systems and Networks**: Implement a distributed system to facilitate communication between various platforms. Consider dependencies and network reliability for seamless interactions.
6. **Security**: Prioritize user data protection and security. Leverage the security capabilities of the recommended platform to safeguard user information across platforms.