

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 2.0

## 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 5**

[**Recommendations 6**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <06/18/23> | <Brent Meraviglia> | <Project Three> |

## [Executive Summary](#_sbfa50wo7nsh)

The goal of this project is to create a web-based version of the game Draw It or Lose It, which is currently an Android-only application. The game involves multiple teams and players trying to guess a puzzle based on progressively rendered stock drawings. We must develop a scalable, efficient, and reliable solution that supports multiple platforms. It must offer an intuitive and engaging user experience. The proposed solution will leverage modern web development techniques to ensure compatibility across different devices and operating systems.

## Requirements

The game should be playable with one or more teams.

Each team should consist of multiple players.

Game and team names should be unique.

Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

The primary constraints for this project are related to compatibility and concurrency. Ensuring the game works seamlessly on multiple platforms and browsers requires rigorous testing and compatibility checks. Also, supporting multiple teams and players simultaneously demands an efficient and robust concurrent processing mechanism. Performance can be a significant constraint, especially when rendering images in real-time for multiple users. Lastly, we need to ensure unique naming for games and teams and maintain only one instance of a game in memory at a time, which adds complexity to our design.

## [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 main classes are GameService, Game, Team, and Player, all of which inherit from the Entity class. GameService is a singleton class that manages the creation and retrieval of games. Each game can have multiple teams, and each team can have multiple players.

The diagram depicts several object-oriented principles. Inheritance is used, where Game, Team, and Player classes inherit from the Entity class. Encapsulation is also used, where attributes are private, and access is controlled through getters and setters. Polymorphism is shown as all the child classes of Entity override the toString() method. Lastly, Singleton design pattern is employed in GameService to ensure only one instance of this class exists in memory.

**"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** | (Secure but costly)  Mac OS is reliable and secure but may not be the most cost-effective option for hosting a web-based application due to its high cost and hardware restrictions. | (Stable, secure, cost-effective)  Linux is an excellent choice for server-side hosting of web applications due to its stability, security, open-source nature, and cost-effectiveness. | (Compatible but costly)  Windows offers great compatibility and support, but it may not be as cost-effective or flexible as Linux. | (Not ideal due to resource limits)  Hosting a web-based application on mobile devices is not ideal due to limited resources and scalability issues. |
| **Client Side** | (Costly, but significant market share)  Mac users represent a significant market share, and supporting Mac clients is necessary. However, developers may need to invest in Mac hardware, increasing development costs. | (Lower cost, smaller market share)  Linux clients represent a smaller market share. However, supporting them helps ensure accessibility and inclusivity. The cost implications are typically lower as Linux-based development tools are mostly open-source. | (Cost varies, large market share)  Windows has the most substantial market share among desktop operating systems, making it vital to support. Development costs vary, but the widespread use of the platform may reduce compatibility issues. | (Crucial due to widespread use, higher development costs)  Supporting mobile devices is crucial given the widespread use of smartphones. Development costs can be higher due to the need for mobile-specific development and testing tools. |
| **Development Tools** | Tools: Xcode and Visual Studio  Languages: Swift, Objective-C, and cross-platform languages like Java. | Tools: Eclipse, NetBeans, and IntelliJ IDEA.  Languages: C, Python, Java, JavaScript. | Tools: Visual Studio, PyCharm, and IntelliJ IDEA.  Languages: C#, .NET, Python, Java, JavaScript. | Tools: Android Studio, Xcode, Xamarin, React Native.  Languages: Java, Kotlin, Swift, Objective-C. |

## Recommendations

1. **Operating Platform**: I recommend using the Linux platform. It allows for broad and cost-effective expansion due to its open-source nature and support for multiple environments.
2. **Operating Systems Architectures**: Linux is a monolithic kernel-based OS. It provides a comprehensive functionality set within the kernel. It's efficient and allows for direct hardware manipulation when necessary.
3. **Storage Management**: MySQL could be used. This system works well with Linux and can handle complex queries and large datasets.
4. **Memory Management**: Linux uses a demand paging system which reduces the amount of wasted memory by only loading pages when needed. Additionally, Linux uses swapping to free up memory, moving less frequently used data from RAM to disk space. This allows Linux to effectively handle the memory requirements of the Draw It or Lose It software.
5. **Distributed Systems and Networks**: In a distributed system, Draw It or Lose It can leverage networking protocols like TCP/IP and HTTP to ensure reliable communication between various platforms. We can use technologies such as JSON or XML for structuring data. Connectivity and outage concerns can be addressed through fault-tolerant design, load balancing, and health checks. Systems like Kubernetes can automate this, ensuring uptime and reliable service.
6. **Security**: Keeping user information safe is very important. On Linux, we can use security tools like SELinux or AppArmor to help us do that. To protect data while it's being sent between different devices, we can use things like TLS/SSL encryption. To keep stored data (like passwords) safe, we can use complex methods to disguise them, and encrypt sensitive data. It's also very important to keep checking our security measures and updating them. Lastly, we’d need to follow privacy laws like GDPR or CCPA, which tell us what we can and can't do with user data. Following these laws not only keeps us out of legal trouble but also helps users trust us more.