

Draw it or Lose it

# **CS 230 Project Software Design**

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 | 03/20/2025 | Megan Jones | * A game will have the ability to have one or more teams involved. * Each team will have multiple players assigned to it. * Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. * Only one instance of the game can exist in memory at any given time. |

## [Executive Summary](#_sbfa50wo7nsh)

Currently, Draw It or Lose It is an Android-only application. The proposed idea is to move forward with the game as a web-based environment. The solution is to create a Java application that will include classes such as GameService, Game, Team, and Player. These classes will interact to manage the games, teams as well as the players. The Entity class will provide a base for all the entities with attributes such as ID and name. Should we move forward with this design approach, we will meet the software requirements and create a successful new way to enjoy this game application.

## Requirements

The game will be a web-based distributed environment using the Java Programming Language. It’ll be an object-oriented design and must meet scalability and performance requirements due to a growing user interface.

## [Design Constraints](#_2et92p0)

**Web-Based Distributed Environment:** This new way to enjoy the classic Game application will be through a web-based distributed environment. The application should be accessible through the internet or internet browser and support multiple users at the same time. The application needs to be designed with network latency and security considerations in mind.

**Java Programming Language:** The software will be developed using Java Programming Language, this will cause constraint limits as the technology stacks Java-based frameworks, libraries, and tools. It will also adhere to Java coding conventions and best practices.

**Object-Oriented Design:** The application will follow object-oriented design principles. This will promote modularity, reusability, and maintainability. The classes will be designed with proper encapsulation, inheritance, and polymorphism. The codebase will be flexible and extensible to help with applying design patterns and abstraction techniques.

**Scalability and Performance:** The game application will have growing numbers of games, teams, and players; therefore, it should be scalable to accommodate the increasing user base without significant performance degradation. This will require optimizing database access, considering caching mechanisms, and also looking at minimizing resource consumption.

**User Interface:** The game application will be an engaging experience for the players using an interface that is intuitive, user-friendly, and responsive. The necessities when designing this responsive and visually appealing user interface, such as HTML, CSS, and JavaScript, will allow the players to interact with the games, teams, and players efficiently.

## [Domain Model](#_8h2ehzxfam4o)

Below you can see The Gaming Room UML Diagram that represents the domain model of the game application. It includes classes such as GameService, Game, Team, and Player. These are shown as interconnected to fulfilling the software requirements through the Entity class, as this serves as the base class for all entities in the application. This demonstrates the principle of inheritance, allowing other classes to inherit these attributes and behaviors. Using the Entity Class allows us to avoid code duplication and ensure consistency across entities.

The GameService class manages the games. List of games, methods to add to games, the ability to retrieve games by ID or name, and retrieving the total game count. Following the Singleton design pattern, this class ensures a single instance is accessible throughout the game application. SingletonTester class “uses” the ProgramDriver class to test this behavior.

The Game class represents a game in the application. The list of teams and methods to add to the teams are maintained and provided using this class. You can also retrieve a string representation of the game. The Team class maintains a list of the players, provides methods to add players, retrieval a string representation of a team, and allows the user to add players. The Player class provides methods to retrieve a string representation of individual players.

Collectively these classes demonstrate the principle of composition, complex objects are built by combining simpler objects. This composition creates a hierarchical structure to manage games, teams and players efficiently. The Game class consists of teams and the Team class consists of players.

**"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** | Characteristics:  -Unix-based environment  Advantages:  -Robust security and stability  -Apache web server and PHP support features  Weaknesses:  -Limited hardware  -Increased pricing | Characteristics:  -Top choice for web-based software applications  -Large range of distributions with high stability and scalability  Advantages:  -Cost-effective  -Customizable  -Known for Security and reliability  Weaknesses:  -More technical expertise to set up and manage | Characteristics:  -Common platform for web applications  -Applications can be built using Microsoft technology  Advantages:  -Easily integrated with Microsoft tools and technologies  -Large range of web server options like IIS  -Good support for .NET applications  Weaknesses:  -High-cost licensing  -Not well-suited for open-source software development | Characteristics:  -Typically doesn’t host web applications  -Servers support mobile app data  -All mentioned platforms can be used as backends  Advantages:  -Reduced latency due to data proximity  -Cost-effective for special use cases  -Small-scale applications are easy to deploy  Weaknesses:  -Limited resources can cause performance issues  -Scalability challenges  -Battery drains and hardware failures cause reliability issues  -potential vulnerabilities cause security risks  -Bandwidth limits and Network dependent  -Complex maintenance and management |
| **Client Side** | Software Development Considerations:  -Limited to Xcode, Apple’s development tool.  -Higher Costs and an understanding of Swift and Objective-C could be required. | Software Development Considerations:  -Development varies upon distribution and desktop environment  -Lower Costs unless expertise is required for distribution-specific considerations | Software Development Considerations:  -Requires Visual Studio for Windows clients  -Cost varies  -Experts in .NET languages may be required | Software Development Considerations:  -Platform-specific development  -Costs vary on the number of platforms  -Swift, Kotlin, and cross-platform tool experts may be required |
| **Development Tools** | Relevant Programming Languages and Tools:  -Xcode is the primary IDE for Mac applications  -Languages include Swift, Objective-C, and C++. | Relevant Programming Languages and Tools:  -Large range of programming languages  -Top IDEs are Visual Studio Code, IntelliJ IDEA, and Eclipse | Relevant Programming Languages and Tools:  -Visual Studio is the primary IDE for development  -Supports C#, C++ and more. | Relevant Programming Languages and Tools:  -Xcode, Swift/Objective-C  -Android Studio and Kotlin/Java  -Flutter |

## 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 or Windows
2. **Operating Systems Architectures**:

Linux: The server is excellent for web hosting and customization. The client varies by distribution but is low-cost. Tools support multiple languages, but Visual Studio Code is the most used.

Windows: Good for hosting and strong Microsoft integration. The client requires Visual Studio and experts in .NET. Tools need Visual Studio for development

1. **Storage Management**: Cloud-based database system.
2. **Memory Management**: Windows has paging, virtual memory, and segmentation, allowing data to switch between memory and disk as required.
3. **Distributed Systems and Networks**: Kubernetes distributes architecture through containers across multiple servers.
4. **Security**: Security is strong through being multi-layered and works together from data in to data out.