

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.2

## 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 |
| --- | --- | --- | --- |
| 1.2 | 8/17/2025 | Josiah Bawden | Edits to Recommendations |
| 1.1 | 7/29/2025 | Josiah Bawden | Edits to Evaluation, Recommendations and design constraints |
| 1.0 | 7/14/2025 | Josiah Bawden | Original document, no changes as of yet |

## [Executive Summary](#_sbfa50wo7nsh)

The client is having issues with the multiplayer environment of the game they wish to produce. We will produce the development of a program that verifies the authenticity of any number of players, teams, and game saves. This will be the foundation of the game but will be separate from the gameplay and be primarily backend processing.

## Requirements

* *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. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_2et92p0)

This has been asked to be done in Java which is compatible with web deployment. The current game is on android only and thus Java is a prime contender in adapting it to the PC operating systems.

## [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 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.**

**Starting with our Main(), this resides in the ProgramDriver class which runs the rest of the classes and verifies operations. This directly correlates with the SingletonTester which will be the processor for making sure each game is running on a separate server.**

**Once Program Driver starts running, GameService will start running and thus addGame will be called. The name of the game will be cross referenced with the <List> to prevent numerous games from being iterative. A similar action will be done with each team and player name respectively.**

**The three classes are subclasses of Entity. Entity consists of Id and name in its protected attributes. The default constructor -Entity is blocked from being created and must be overloaded with other constructors that are apart of the diagram.**

## 

## [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** | The Apple servers are beneficial for integration within the Apple ecosystem. Unfortunately, they are also significantly more expensive with the monopoly on hardware and with the current company performing work on android (a linux kernel) this could prove to be a difficult and expensive endeavor. | Linux is based on high adaptability and an open source nature. It is extremely cost effective with practically any level of processors being able to run the numerous software’s and it is compatible with many web technologies. It does not have automatic updates unlike Windows and MacOS and there is a steeper learning curve. Though this should not be an issues considering we are a software company. | This is one of the most familiar web services with a strong support for .NET based web applications. It has high security features that are streamlined within the Microsoft ecosystem. You are limited by Windows architecture but this is of minimal disturbance based on what we are trying to accomplish on the server side. | Mobile is not very well adapted for server work. They excel in portability but that is sadly where the positives end. They have limited processing power and numerous security vulnerabilities due to this. Although they are useful for clients connecting to the servers. |
| **Client Side** | Supporting multiple types of clients on Mac requires consideration of the platform’s tightly controlled ecosystem. Development costs can be high due to the need for Mac specific hardware and software licenses, such as Xcode. Time investment is moderate, as Apple’s environment simplifies development for iOS and macOS clients, but cross-platform compatibility (e.g., with Android) demands additional frameworks. | Linux’s open source nature makes it cost effective for supporting diverse clients, as it runs on varied hardware without licensing fees. However, development time can be longer due to the fragmented ecosystem, with multiple distributions requiring testing for compatibility. Expertise in languages like Python, C++, or Java is essential, and our team’s familiarity with Android’s Linux kernel gives us an edge in adapting to Linux client environments. | Windows client side development benefits from its widespread use and familiarity, reducing training costs and time for developers. Supporting multiple client types is streamlined by tools like Visual Studio, which integrate well with Windows ecosystems. However, costs arise from licensing fees for Windows and associated tools, though these are often offset by the platform’s robust support for enterprise applications. Expertise in C#, .NET, or JavaScript is required, and our team’s software development experience aligns well with these needs. | Mobile devices, particularly Android and iOS, are critical for client side applications due to their ubiquity. Development costs are moderate, as tools like Android Studio (free) and Xcode (Mac) are accessible, but supporting both platforms increases expenses due to dual development pipelines. Time investment is significant, as optimizing for diverse device specs (especially Android’s fragmentation) and ensuring security across mobile networks is complex. |
| **Development Tools** | For deploying software on Mac, Swift is the primary programming language for native macOS and iOS applications, with Objective-C still relevant for legacy support. Xcode is the go to IDE, offering robust tools for development, debugging, and testing within the Apple ecosystem, but it requires Mac hardware, increasing costs. Jetbrains offers the best variety of IDE’s for our needs but the cost on them is steep with almost $800 per user per year. | Linux development relies upon tools that tailored to their ecosystem. Languages like Python, C++ and Java are all well established within the operating system variations. For IDEs, some useful ones are Visual Studio, IntelliJ IDEA or Eclipse. Jetbrains offers the best variety of IDE’s for our needs but the cost on them is steep with almost $800 per user per year. | Windows development leverages C# and .NET for native applications, with JavaScript and TypeScript for web-based clients. Visual Studio is the flagship IDE, providing comprehensive tools for coding, debugging, and testing, particularly for .NET-based applications. Jetbrains offers the best variety of IDE’s for our needs but the cost on them is steep with almost $800 per user per year. | Mobile development primarily uses Kotlin and Java for Android, and Swift for iOS, with cross platform frameworks like Flutter or React Native for unified codebases. Android Studio is the standard IDE for Android, while Xcode is required for iOS, tying development to Mac hardware. Visual Studio Code supports cross-platform frameworks, enhancing flexibility. Jetbrains offers the best variety of IDE’s for our needs but the cost on them is steep with almost $800 per user per year. |

## 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 would argue that Linux is the best operating system for The Gaming Room to configure for their own needs. Considering the company is currently mobile based, Linux would adapt well to moving to the desktop ecosystem with relatively easy cross platform compatibility to Windows and all Linux OS’s.
2. **Operating Systems Architectures**:   
   The Linux kernel, similar to Android’s, handles core functions like process management, memory allocation, and networking, with user-space tools enabling flexible service management. For clients, Android uses a modified Linux kernel with a layered architecture, optimized for mobile devices. Linux’s architecture supports containerization (Kubernetes) for scalable game servers, and its compatibility with Android ensures efficient integration with mobile clients.
3. **Storage Management**: It is recommended to USE Zettabyte File System for Linus server adaptation. It has features like data integrity checks, snapshots and compression that verify reliable storage for game data. It has high scalability, is open source and can support cross platform with Windows.
4. **Memory Management**:   
   Linux’s memory management relies on efficiently allocating resources for game servers. For "Draw It or Lose It," Linux employs virtual memory to handle game processes like real time multiplayer sessions and leaderboard updates. Tools like cgroups ensure that memory intensive tasks, such as rendering or user data processing, don’t overwhelm the system. Considering our mobile based expertise with Android’s Linux kernel, this approach aligns with our existing skills, and Linux’s memory efficiency supports cross-platform clients on Windows and other Linux distributions with minimal overhead.
5. **Distributed Systems and Networks**:   
   To allow "Draw It or Lose It" to communicate across various platforms, a client server architecture using WebSocket connections is recommended, hosted on Linux servers. Cross platform frameworks like Flutter ensure consistent gameplay on Android, Windows, and other Linux OS’s. Kubernetes manages server scalability, while load balancers mitigate network outages. Considering our Android background, Linux’s compatibility simplifies server client integration, ensuring smooth cross platform performance.
6. **Security**:  
    Security is a must have for protecting user information on and between platforms. Linux servers leverage SELinux to enforce strict access controls, safeguarding game data like user profiles and scores. TLS/SSL encryption secures data transmission to clients on Android, Windows. Android’s Keystore and sandboxing protect mobile client data, and Linux’s open-source nature allows regular security patches, aligning with the mobile expertise of The Gaming Room.