

<Name-of-Software-Application>

# **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 |
| --- | --- | --- | --- |
| 1.0 | 08/11/1993 | Ryan Davis |  |

**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)

It sounds to me the issue is the staff at the gaming room does not know how to code? Or maybe it is just the environment. The gaming Room wants to build a android game app Draw It or Lose It based off a game in the 80’s, This application will render images instead of contestants themselves.

## Requirements

Team Player Support: allow one or more teams with multiple players

Unique Team and Game Names: game names must be unique.

Game instance Management: only one game can exist in memory at once, utilizing unique id’s should prevent this.

## [Design Constraints](#_2et92p0)

* Scalability: must support high volume of users playing simultaneously on different OS. Recommended backend database to handle this
* Data Consistency: synchronization between OS is critical
* Security: With all apps I always recommend some form of security
* Single Game Instaance: only one game instance at a time
* Cross platform: windows, mac, android, ios all can have similar but different infastructure needed to be accounted for.

## [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)

Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.

Game , Team and Player classes are have the unique identifier attribute to ensure only one instance runs at a time. The Game class is the core of the diagram representing One to many to the team and game service classes, it controls the flow. The Team class ensures at least one team participates and also demonstrates One to many relationships with the other two mentioned. The player class ensures each player is assigned to one team at a given time and a one to many relationship effectively manages the teams helping with functionality.

OOP: Encapsulation: the three mentioned above encapsulates its own data giving more control over those three data points.

Inheritance: I’m sure inheritance will be used as it just makes everyone’s life easier, writing and managing over long term.

SRP: each class has a well defined responsibility or role this helps with scalability and maintnance.

"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 can handle server tasks but is rarely used for large-scale hosting due to high costs and limited server options. It’s better for apps that are deployed elsewhere, like in the cloud. | Linux is ideal for servers due to its stability, security, and scalability. It’s popular for high-traffic web apps and cloud services but can be tough to set up for beginners | Windows servers are great for enterprise apps, especially with .NET and SQL Server integration, but it's less used for Unix-based services. | Mobile apps rely on external servers, often using cloud services like AWS or Firebase for performance. Optimizing back-end is crucial. |
| **Client Side** | Mac is perfect for iOS/macOS development with seamless hardware/software integration. It’s secure and ideal for Apple ecosystem apps. | Linux offers great flexibility for developers but isn’t the easiest for non-tech users. It's perfect for open-source projects and coding-focused environments. | Windows dominates desktop app development, especially with C# and .NET. It’s versatile but not as optimized for web dev as Mac or Linux. | iOS apps are built with Swift/Xcode, and Android apps with Kotlin/Android Studio. Cross-platform tools like React Native and Flutter save time. |
| **Development Tools** | Xcode is key for iOS/macOS development. Mac also supports React Native, Docker, Homebrew, and Node.js, making it versatile for web and mobile devs. | Tools like VSCode, IntelliJ, Docker, and Git are widely used. Linux is customizable, but getting it set up may take some extra time. | Popular tools include Visual Studio, SQL Server Management Studio, and VSCode. Docker works, and WSL allows Linux tools. Azure integration is smooth for cloud work. | Android studio is used often, xcode for IOS, there are cross platforms as well.Android Studio for Android, Xcode for iOS, and cross-platform tools like React Native and Flutter are essential. Firebase often handles backend services. |

## Recommendations

Analyze the characteristics and techniques specific to various system architectures and make a recommendation to The Gaming Room:

1. **Operating Platform**: I recommend Linux because it’s an open-source operating system, known for its stability, scalability, and flexibility. As an open-source platform, Linux allows for high levels of customization, which can be tailored to fit the specific needs of your game, Draw It or Lose It. It’s commonly used in enterprise environments because of its ability to handle large-scale operations with reliability. For The Gaming Room, Linux would provide a solid server-side foundation, capable of managing increasing numbers of players and expanding into new gaming environments. Additionally, Linux has a large support community and an extensive range of tools, which can be leveraged for optimization and problem-solving.
2. **Operating Systems Architectures**: The Linux operating system supports two main architectures: x86\_64 and ARM. The x86\_64 architecture is known for its ability to support high-performance servers, which is ideal for hosting a large number of players and maintaining the smooth operation of the game. This architecture allows for excellent scalability, meaning that as more users join, the system can handle the load without performance degradation. On the other hand, ARM architecture is specifically designed for mobile devices. It consumes significantly less power while maintaining compatibility with mobile operating systems such as iOS and Android. This makes it the ideal architecture for expanding Draw It or Lose It to mobile platforms, ensuring that mobile gamers can enjoy the same quality experience while minimizing battery usage.
3. **Storage Management**: For handling game data, I recommend using a combination of Amazon S3 and PostgreSQL. Amazon S3 is perfect for storing large amounts of static content, such as game assets, images, and videos. It’s a cloud-based storage solution that is scalable, which means it can grow as the game expands without the need for major changes to the infrastructure. PostgreSQL, a relational database management system, is a great fit for managing dynamic, structured data such as user profiles, game progress, and statistics. By combining these two storage solutions, The Gaming Room can efficiently manage both static and dynamic data. Amazon S3 will handle the heavy storage needs for assets, while PostgreSQL will manage user data and interactions.
4. **Memory Management**: The memory management capabilities of Linux are well-suited to handle the resource-intensive nature of games. Linux uses virtual memory, which means it only loads the necessary parts of an application into memory. This ensures that the server is not overloaded with unnecessary processes, optimizing performance. For Draw It or Lose It, this would mean smoother gameplay, even when the player base grows or when there are spikes in activity. By managing memory efficiently, Linux ensures that large applications and games do not run into performance bottlenecks, which is essential for maintaining a quality gaming experience.
5. **Distributed Systems and Networks**: To allow Draw It or Lose It to operate across various platforms and ensure seamless communication between them, a distributed system is necessary. The game’s components will need to communicate in real time, which can be accomplished using RESTful APIs or gRPC. These communication protocols are well-suited for multiplayer games where real-time synchronization of data is critical. Additionally, using distributed databases ensures that the game remains functional even during connectivity issues or partial outages. Cloud platforms like AWS or Google Cloud can provide the necessary infrastructure to keep Draw It or Lose It running smoothly across multiple devices and environments, allowing The Gaming Room to expand their game onto different platforms without compromising performance.
6. **Security**: Security is imperative when handling sensitive user data, especially in a distributed environment. To protect user information, I recommend implementing end-to-end encryption for all communication between servers and clients. This will ensure that data transmitted over the network remains secure. Additionally, encryption at rest should be applied to stored data, so even if the storage system is compromised, the data remains unreadable. Firewalls and intrusion detection systems will add extra layers of security, protecting the game from unauthorized access or attacks. Role-based access control (RBAC) should also be implemented, ensuring that only authorized personnel can access sensitive areas of the system. To further enhance security, the use of secure communication protocols such as HTTPS and TLS will help protect the integrity of data being transferred. For user authentication, implementing a framework such as OAuth 2.0 will provide a secure way for users to log in and protect their credentials from potential breaches.