

The Gaming Room

# **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 | 09/20/2024 | Sergio Irianda | The cover page, executive summary, requirements, design constraints, domain model, and recommendations have all been updated. |
| 1.0 | 10/05/2024 | Sergio Irianda | Design constraints, domain model, and evaluation have been updated. |
| 1.0 | 10/15/2024 | Sergio Irianda | Recommendations has been updated. |

**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)

The Gaming Room Project aims to develop a web-based game that serves multiple platforms to expand the game’s reach beyond Android. The key requirements of the client are enabling multi-team support, ensuring unique identifiers to prevent conflicts, and managing single instance games. A singleton design pattern will be implemented for object creation to ensure that only one instance of the game exists, while an iterator pattern will help manage and prevent conflicts between teams and their members. The development of this project will adhere to industry standards and best practices to ensure scalability and deliver a functional code.

## 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)

* The application's security setup will need to be modified. How users log in on mobile devices will depend on the device's security features. However, the game will need better security and login methods to verify users and keep their information safe in a client-server system.
* The game is currently an Android app, and to extend its availability to other platforms, we will have to convert it into a web application that can be accessed from various operating systems. This can create compatibility issues.

## [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 UML class diagram shows a visual representation of the game design. The entity class has a linked relationship with the game, team, and player classes. The arrow indicates attribute inheritance form all these classes from the parent class, being the entity class. On the top left corner, the programDriver Class is using direct association by pointing to the singletonTester. This means that he programDriver will use the SingletonTester to test the code to ensure the restriction of having one instance of a game. The Gameservice class ensures the structure and functionality of the game. The lines connecting

each class tell us that they all are associated. The numbers tell us the number of associations within each class. The diagram shows us it fulfills the needs of having a unique game, teams and

player by creating a class for each.

The application includes a main driver class that starts creating games, teams, and players. The actual creation happens through the GameService class, which uses a singleton design pattern to ensure that only one instance of the GameService class exists in memory at any time. The entity object is used as a base for other classes to inherit from, we ensure that each class has a unique identifier and name. The Entity class has two protected attributes: id and name. The default constructor is also protected, which stops null objects from being created, so only the other constructors can be used. The iterator pattern ensures that the game will have multiple teams and players. Polymorphism and inheritance are used to extend the Entity class and create different constructors. Encapsulation and abstraction are also used when adding teams.

**"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** | Has a great interface. Unix-based, known for stability and performance. Seamless integration with other Apple services. Limited server software options compared to others. Licenses and hardware are expensive. Not great with customization. | Free, open source, and customizable. No licensing costs, strong community support, and scalability. Needs technical expertise for setup and management. The interface can be difficult for new users. | Easy to use and set up. Extensive software support. Licensing costs can be high. Uses Windows server. | Doesn't possess the same power as other operating systems. Requires separate development environments. Large user base and community for gaming. Relies on back-end servers.  Fees for app store deployment. |
| **Client Side** | Expensive for users. Limited customization. Very good security. The application should function in Safari and other browsers available on macOS. Compatibility with macOS's security features. | The application must be compatible with common web browsers (e.g., Firefox, Chrome). The design ensures usability across different screen sizes and resolutions. Testing must be thorough, since Linux users may use different desktop environments and browsers. Needs to be easily accessed via standard web browsers without additional configuration. | The application must be compatible with common web browsers like Edge, Chrome, and Firefox. Needs multiple languages and frameworks. A lot of security updates. Design should account for different Windows devices, including desktops and tablets. Different versions of Windows can have an impact on browser performance and capabilities. | Easy to use. Must be functional on both iOS and Android devices. The design needs to cater to touch interactions and smaller screens. Developers need to account for different screen sizes, resolutions, and performance specifications. |
| **Development Tools** | Supports most mainstream languages and IDEs, though some tools may require licenses. Specialized skills might be needed based on the chosen tools, potentially requiring multiple development teams. Supports Swift, Objective-C, JavaScript, Ruby, Python, Xcode), Visual Studio Code, Sublime Text, Atom, and Git (GitHub, GitLab, Bitbucket). | Supports most mainstream languages and IDEs, though some tools may require licenses. Specialized skills might be needed based on the chosen tools, potentially requiring multiple development teams. Supports Python, Ruby, Java, PHP, JavaScript (Node.js), Visual Studio Code, Atom, Eclipse, PyCharm, NetBeans, and Git (GitHub, GitLab, Bitbucket). | Supports most mainstream languages and IDEs, though some tools may require licenses. Specialized skills might be needed based on the chosen tools, potentially requiring multiple development teams. Supports C#, .NET, Java, JavaScript, Python, Visual Studio, Visual Studio Code, JetBrains Rider, Eclipse, Microsoft Azure, and Git (GitHub, Azure DevOps) | Supports most mainstream languages and IDEs, though some tools may require licenses. Specialized skills might be needed based on the chosen tools, potentially requiring multiple development teams.  . Supports Swift (iOS), Kotlin (Android), Java (Android), JavaScript (React Native, Flutter), Xcode (for iOS), Android Studio (for Android), Visual Studio Code (for cross-platform), Git (GitHub, GitLab, Bitbucket). |

## 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 the Linux operating system for The Gaming Room’s expansion of *Draw It or Lose It*. Linux is open source, has lower cost, has abundant development tools, and it’s very flexible. In particular, the Ubuntu Server will provide the reliability, security, and performance the game needs.
2. **Operating Systems Architectures**: Linux architectures are based on the hardware, kernel, and shell. The key pieces of hardware are made of the memory and CPU. This provides the processing and power necessary for the game. The kernel is a bridge between applications and the actual data processing in the hardware. Shell is software that provides an interface for users to the operating systems. In addition, Linux supports various architectures such as ARM making it easier to expand to mobile systems.
3. **Storage Management**: An ideal file system for The Gaming Room is Ceph. It’s a free open source system and can facilitate a lot of data across many servers while maintaining redundancy. This file system allows the scaling of both file-based and object storage. This flexibility is what *Draw it or Lose it* needs as the game expands and user data increases.
4. **Memory Management**: Ubuntu Server is very effective at managing memory, which is crucial for *Draw It or Lose It*. It uses virtual memory, allowing the game to use more than the physically available memory. It also uses dynamic memory allocation, which can help manage memory while the game is running. In addition, it compresses inactive memory, maintaining performance high by freeing up resources for active processes. All these different memory management techniques allow for a smooth experience of the game.
5. **Distributed Systems and Networks**: We recommend using a microservices architecture for the game work across different platforms. This means the game can be split into different projects, that can be developed and scaled separately. For communication, the use of RESTful APIs for standard requests and WebSockets for real-time interactions will create the player experience the game is looking for. Load balancers will distribute traffic evenly, ensuring availability to users even during peak times. In case of outages, a backup system will be in place keeping everything running smoothly.

1. **Security** : We recommend using SSL/TLS encryption for all data transmission to keep data safe. For user authentication, OAuth 2.0 will ensure secure access across all platforms. Regular security assessments and updates will help identify and mitigate potential vulnerabilities. Role-based access control (RBAC) will also restrict access to sensitive game data and further enhance security posture.