

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  1.1  2.0 | 03/20/25  04/04/25  04/18/25 | Charles Mills  Charles Mills  Charles Mills | First version – no revisions  Evaluated various operating platforms  Gave final technical recommendations for application |

**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 is launching a web-based version of their Android game *Draw It or Lose It,* based on a TV show called *Win, Lose or Draw*. The client requires that the web-based application supports multiple teams with multiple players, ensure that all names (games, teams, players) are unique, and limit the system to one active game instance at a time. To meet the clients needs, we propose a software design that uses the principles of object-oriented programming such as inheritance, abstraction, and encapsulation. A Singleton class (GameService) will manage all game, team, and players data to ensure a consistent state of gameplay and prevent duplication of data.

## Requirements

Some key business and technical requirements are as follows:

1. Team-based game structure allowing multiple players.
2. Unique identifiers for games, teams, and players.
3. Single instances of games, teams, players.
4. Gameplay that is both engaging and which follows the rules set forth by the client.
5. Scalable on a web-based platform.

## [Design Constraints](#_2et92p0)

Some restraints that have been identified are as follows:

1. Using a Singleton class (GameService): only one instance of a game can exist in memory at any given time. This requires implementing a Singleton design pattern in the GameService class. In a distributed environment, this provides data consistency across sessions and simplifies memory management.
2. Unique game, team, and player IDs: these IDs must be unique across all saved games. Methods to check for pre-existing IDs must be created. This will support scalability and future multi-player game session management.
3. Web-based architecture: the application must run in a web environment, likely in a browser using technologies such as HTML, CSS, and JavaScript. The backend will need to handle data requests from the client’s browser and update the frontend efficiently to keep the game running smoothly.
4. Support for multiple games, teams, players: The application must use an object-oriented design that supports hierarchical relationships, such as a game containing multiple teams, and each team containing multiple players. Structuring the data this way makes it easier for the backend to manage and update nested objects across different sessions and user interactions.

## [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 diagram below shows an application called The Gaming Room. The application is initialized by a main driver class (ProgramDriver) which initiates the game with teams and players by utilizing the GameService class. The GameService class follows a Singleton pattern, ensuring that only one instance of the service exists as to avoid duplication. The creation and management of game instances, teams, and players is centered around the Entity base class, which shares attributes and methods for accessing and updating those attributes.

The program (and the UML diagram) demonstrates some of the pillars of object-oriented programming such as inheritance, encapsulation, and abstraction. The Entity class, through inheritance, provides attributes and methods for the Game, Team, and Player classes, as well as abstraction by obscuring the implementation details of the class while allowing others to interact and share common functionality. The GameService class uses encapsulation to keep List<Game> safe from tampering while providing the methods necessary to retrieve and modify the information it contains.

**"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** | MacOS is a user friendly and secure operating system based on Unix. The OS is well integrated with Apples larger ecosystem and supports web technologies such as Node.js, Ruby on Rails, and Python. Mac is not widely used in server environments due to the higher cost of hardware, making scaling very expensive. | Linux is known for being cost-effective and secure, an ideal platform for hosting web servers. Linux servers are highly scalable and have extensive support for open-source tools as well as native support for containerization which further allows it to be used for managing larger, more complex distributed systems. Linux does, however, have a bit steeper of a learning curve than Mac or Windows, and could be less familiar to many developers. | Windows is a great OS for hosting webservers using tools such as Internet Information Services or Apache. While Windows servers are very common in enterprise settings, they come with high licensing costs and require a higher degree of configuration to match the performance and scalability of Linux. Windows does however provide a familiar interface and integration with other Windows and Microsoft tools that developers may find useful and that may make this a good choice for some organizations. | Mobile devices are first and foremost a personal, client-side platform and are not used for hosting web applications. These devices are mainly used for consumption, and not development. Although they are not meant to be used for development or deployment of software, they are a crucial part of the user experience today and are widely developed for. |
| **Client Side** | Mac offers a robust and tightly integrated environment that some prefer to develop in and consider more fluid and efficient, however I reiterate the high cost of hardware as well as a certain degree of technical knowledge that may be necessary due to the MacOS being a less popular development environment than, say, Windows or Linux. | On the client end, Linux supports web development through a wide array of open-source tools. Developers can use a range of IDEs and robust package management systems for installing dependencies. Linux is the most cost-effective of the OSs, as it is free, and supports many free tools and software that also speed up development, but the OS itself may require some learning from the developer to make the most of it. | Windows offers a familiar interface for client-side development, making it accessible to more developers than Mac or Linux. The Windows ecosystem supports a wide range of IDEs and is generally considered a highly productive environment that is available for a relatively low cost. While there is a high level of familiarity and efficiency, some web development tasks may require a higher level of configuration than that of Linux, such as command line tools. | As a purely client-side device, the OS is made for convenience and efficiency. Developers have a wide range of options for developing and testing on, so price can vary greatly. Developing for these devices can be time consuming, as there are a variety of OS and hardware considerations that must be properly developed for and tested. Mobile development also requires a range of specialized skills. For native app development the developer must have experience in using languages supported by iOS and Android. |
| **Development Tools** | Xcode is Apples IDE for macOS and iOS applications. Using this for client-side development for iOS and macOS apps offers the advantage of native support and tight integration which can speed up the development process. Many developers also use Visual Studio Code (as VS is not supported for Mac anymore). The macOS terminal, coupled with kits like HomeBrew also offers powerful tools to developers due to its Unix-based nature. macOS supports all major browser with built-in developer tools and has Git integration in most IDEs for easy version control. | Visual Studio Code, Vim, Emacs, and many others are highly configurable and supported by Linux. Languages like Python, Ruby on Rails, PHP, and JavaScript are popular on Linux, and are extensively supported by the development community. Bash and Zsh are powerful command line tools that are supported, and Git is standard and well-integrated. Apache and Nginx are widely used for configuring web servers. | Visual Studio and Visual Studio Code are common IDEs for Windows development, but many others are also supported. It supports a wide range of programming languages such as C#, Python, Java, C++, and .NET languages. WSL (Windows Subsystem for Linux) is another powerful tool available that allows developers to run a Linux environment on Windows. Windows also provides robust command-line tools for scripting, automating, and system management. | The tools used for Mobile development vary from OS to OS. For iOS devices developers use Xcode, and for Android there is Android Studio. There are also cross-platform solutions such as React Native and Flutter which enable development for both platforms from a single codebase. Both Xcode and Android Studio include built-in emulators so that developers can test their applications on various devices with different configurations without the need for many different pieces of hardware. |

## 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**: Based on the evaluation I gave earlier, I would recommend that The Gaming Room use Linux as the operating system for the server hosting Draw It or Lose It. Linux is commonly used for servers because of its low-cost, scalability, and that it works well with a lot of web-based tools. It also supports things like containerization, which can help as the game grows and needs to support more users or platforms. While it might take a bit of getting used to for some, it gives the most flexibility overall.
2. **Operating Systems Architectures**: Linux runs on a monolithic kernel architecture, meaning everything works together in one space, which helps with speed and efficiency. It’s a good fit for running large-scale apps like this game. It’s also well-supported by cloud services and works with tools like virtual machines and containers, which could be useful if the game needs to scale up or run in different environments.
3. **Storage Management**: Linux supports reliable file systems like ext4, which are great for handling lots of data. It includes features like journaling, which can help with recovery if anything crashes. For this game, we’ll need to store things like player data, game states, and drawings. A combination of ext4 and a relational database like MySQL would be a good setup for organizing and backing up that kind of data.
4. **Memory Management**: Linux handles memory with tools like virtual memory, paging, and swapping, which help prevent crashes when under load. For Draw It or Lose It, where multiple people could be playing at once, the server needs to manage several active sessions. Linux can allocate memory to active processes while moving other data to swap space to keep things running smoothly and not become overloaded. Tools like caching and session management will also help with performance.
5. **Distributed Systems and Networks**: Since the game needs to work across platforms (like web and mobile), we’ll want to use a distributed setup. This means the app is split into different parts that work together, like a web or mobile frontend talking to a backend server. They would connect over the internet using protocols like HTTP and WebSockets. The system should also be set up to handle things like connection issues or outages, using tools like load balancers and monitoring software to keep everything working across devices.
6. **Security**: Security is a must, especially with people’s info being sent back and forth between devices. On the server side, things like passwords and personal data should be encrypted and stored safely. All traffic should use HTTPS so that it’s encrypted in transit. We should also have logins, user roles, and other protections in place to stop unauthorized access. On top of that, using secure coding practices, updating software, and watching for attacks like SQL injection or XSS will help keep the platform safe. I would also place emphasis on staff training for phishing attacks that could put our code and user data at risk.