

# Draw It or Lose It

# **CS 230 Project Three**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.1 | 07/28/2025 | Montana Chum | Initial draft with completed Executive Summary, Design Constraints, and Domain Model sections |
| 1.2 | 08/10/2025 | Montana Chum | Project Two updates with revised Evaluation, Requirements, Executive Summary, Design Constraints, Domain Model, and Recommendations |
| 1.3 | 8/21/2025 | Montana Chum | Project Three updates: Completed Recommendations section including operating platform, OS architecture, storage management, memory management, distributed systems, and security |

**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 plans to expand its mobile game, Draw It or Lose It, into a web-based version that works on multiple platforms. The design will support multiple teams and players at the same time and keep names unique for teams and games. The system will allow only one active game instance in memory. The architecture will apply the Singleton pattern to limit the Game Service to one instance and the Iterator pattern to check for duplicate names before adding new teams or games. This approach supports stable performance, easy scaling, and consistent user experience across platforms.

## Requirements

• Allow multiple teams to participate in the game.

• Ensure each team has more than one player.

• Enforce unique names for all teams and games.

• Maintain only one active game instance in memory at any time.

• Provide scalability, responsive layout, and cross-platform compatibility for desktop and mobile.

## [Design Constraints](#_2et92p0)

• Implement the Singleton pattern to guarantee a single GameService instance.

• Use the Iterator pattern to traverse collections and enforce unique names.

• Keep the architecture lightweight, maintainable, and ready for concurrent users without performance loss.

## [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)

• Encapsulation: Class variables remain private with access through getters and setters.

• Inheritance: Entity serves as the base class for Game, Team, and Player to promote reuse.

• Polymorphism: Methods such as toString are overridden to fit each class.

• Singleton: Limits GameService to one instance in the system.

• Iterator: Traverses collections to prevent duplicate names for games and 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** | Suitable for development but uncommon for production hosting. Requires Apple hardware which raises cost. Stable and secure yet less used for large deployments | Most used for production hosting. Free and open source. Stable and efficient for web apps. Works well in all major clouds and scales smoothly. | Strong fit for .NET and Microsoft services. Clear path with IIS. Licensing adds cost and resource use is higher than Linux. | Not used for hosting. Devices act as clients. Focus on smooth use on iOS and Android. |
| **Client Side** | Needs browser testing on macOS. Stable and friendly for Mac users. Ensure layout and media work in Safari and Chrome. | Smaller general audience but important for technical users. Browser support ensures proper function. | Largest desktop audience. Make Windows browser compatibility a priority. | Critical for reach. Handle screen sizes, touch input, and OS versions for iOS and Android. |
| **Development Tools** | IntelliJ IDEA, Eclipse, Maven, Gradle are supported. Xcode is needed only for native Apple builds or device testing. | IntelliJ IDEA, Eclipse, NetBeans, Maven, Gradle, and command line tools work well. Strong fit for Java and scripting. | IntelliJ IDEA, Eclipse, and Visual Studio are available. Visual Studio is ideal for .NET and C#. | Android Studio for Android and Xcode for iOS when needed. For the web client, use responsive design and test on real devices and emulators. |

## 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**:  
   For the Gaming Room, I recommend hosting Draw It or Lose It on a Linux-based server. Linux is affordable, reliable, and supported by all major cloud providers. Another advantage is that .NET is no longer limited to Microsoft servers, it now runs smoothly on Linux and Mac as well. This gives the client flexibility to stay cost-effective without losing compatibility with their existing development tools.
2. **Operating Systems Architectures:**Linux offers modular architecture, which means only the necessary components can be installed for the game environment. This keeps the system lightweight, reduces security risks, and makes it easier to scale as the user base grows. The architecture also supports containerization (like Docker), which simplifies deployment across multiple servers if needed.
3. **Storage Management:**  
   Linux servers can use file systems such as ext4 or XFS, which are proven to handle large amounts of data efficiently. To support growth, the system can integrate with cloud storage solutions for backups and scaling. This ensures the game’s assets and user data are both accessible and secure.
4. **Memory Management:**Linux uses virtual memory, caching, and swap space to balance workloads when traffic is high. These techniques keep the application stable by preventing memory overflow and ensuring smooth performance even if multiple teams and players are active at the same time.
5. **Distributed Systems and Networks:**  
   To allow communication between different platforms, the game can use REST APIs for general data exchange and WebSockets for real-time interactions during gameplay. Load balancers and redundant servers will help maintain uptime, while failover options reduce the impact of outages. This distributed design ensures players have consistent experience whether they connect from desktop, mobile, or web.
6. **Security:**  
   Security is a top priority. Linux supports tools such as SELinux to control permissions and firewalls to block unauthorized access. Regular updates and patches keep the system protected. Data can be encrypted both in transit (using TLS/HTTPS) and at rest (using file system encryption). These measures safeguard user accounts and ensure compliance with industry standards.