

Draw it or lose it

# **CS 230 Module Seven, Project Three**

Version 3.0

## 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 | 02/06/2025 | Joshua D Rodriguez | Document creation. |
| 2.0 | 02/09/2025 | Joshua D Rodriguez | Evaluation section completed. |
| 3.0 | 02/23/2025 | Joshua D Rodriguez | Recommendation section completed. |

**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 working on Draw it or lose it, a multiplayer web-based game that revolves around quick thinking, competition, and teamwork. Unlike traditional drawing games, this version renders premade images, challenging players to guess the word or phrase within a set time. To ensure smooth gameplay, the system needs to manage game instances, teams, and players efficiently, while preventing duplication and ensuring scalability.

## Requirements

To meet these requirements, we’re implementing key software design principles. The singleton pattern will ensure that only one instance of the game management service exists at any given time, preventing conflicts in game data. Additionally, the iterator pattern will streamline searching and retrieving existing game, team, and player information. By structuring the application with object-oriented design, we can create a flexible, scalable system that meets the needs of both the developers and end users.

## [Design Constraints](#_2et92p0)

Developing a web-based, multiplayer game comes with several design challenges. Below are the key constraints and how they will be addressed.

1. **Preventing Duplicate Game Instances:** only one instance of the Game Service class should exist in memory at any given time. To enforce this, the Singleton pattern will be applied, which will restrict game service creation and ensure all game related data is handled in a single controlled instance.
2. **Efficient Data Management:** games, teams, and players must be stored and retrieved efficiently. Instead of using simple loops, the iterator pattern will be used to traverse lists of game objects, making searching the database more structured and scalable.
3. **Scalability and Unique Identifiers:** It is important for instances in the game to have and maintain unique identifiers, to ensure there are no conflicts when users select names. The Entity class will be introduced to manage shared attributes like ID and name, providing a clean inheritance structure that supports growth.
4. **Web-Based Compatibility and Performance:** the game must also be accessible across different devices and browsers without local installation. This means the backend must handle multiple simultaneous requests efficiently while ensuring seamless client to server communication.

## [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)

* **Game:** Represents an active game and holds a list of teams. The addTeam() method ensures unique team names.
* **Team:** Holds a list of players and ensures unique player names with addPlayer().
* **Player:** Represents an individual participant in a team.
* **Game Service (singleton):** Manages all game instances, ensuring only one instance exists in memory at any given time. It prevents duplicate game creation and efficiently searches for existing games using an iterator pattern.
* **Key Object-Oriented Principles:**

1. **Encapsulation -** Each class manages its own data within controlled access.
2. **Inheritance -** Game, Team, and Player inherit from Entity to avoid redundancy.
3. **Singleton Pattern -** Prevents multiple instances of Game Service, maintaining data consistency.
4. **Iterator Pattern -** Enables efficient searching of games without exposing internal list.

**"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 provides a stable environment but is less commonly used for large scale web hosting. Limited server-side deployment options and higher cost makes it less than ideal. | Linux is the industry standard for web servers. It offers strong security, open-source flexibility, and scalability, making it the best option for hosting the game. | Windows Server is widely used in the enterprise world. However, it comes with higher cost due to licensing. It is also more vulnerable when compared to Linux. | Mobile devices aren’t ideal for server hosting, but cloud-based infrastructure can allow mobile users to connect to a Linux or Windows server seamlessly. |
| **Client Side** | Development for a MAC is streamlined through macOS, but costs are high due to proprietary hardware and software. | Linux clients are rare; however, the game can still be made accessible via web browsers. Compatibility issues may be a concern | Windows is the most widely used OS, making it necessary to ensure seamless client support. Development is straightforward due to high compatibility. | Mobile clients are crucial for accessibility. IOS and Android require separate considerations, with different development tools and store policies. |
| **Development Tools** | Uses Xcode, Swift, or cross platform tools like Unity and JavaScript. Native development is costly, but it does ensure quality software. | Common tools include Java, Python, Node.js, and containerization with Docker. Most flexible option for web-based application. | Development can be done using Visual Studio, C#, .NET, and JavaScript. Strong integration with cloud services but has licensing fees. | Mobile app development requires Android Studio for Android and Xcode for IOS. Cross platform tools like React Native can reduce workload. |

## 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 hosting Draw It or Lose It, Linux is the best choice due to its security, scalability, and cost-effectiveness. It is widely used for web-based applications and offers strong community support as well. For client-side compatibility, the game should be accessible through web browsers, ensuring support for Windows, macOS, and mobile platforms.
2. **Operating Systems Architectures**: A Linux based server architecture allows for stability, efficiency, and open-source flexibility. Web applications built on this architecture can run in a cloud environment, ensuring accessibility across multiple devices without the need for direct platforms dependencies.
3. **Storage Management**: A cloud-based database such as MySQL is recommended for structured game data management. Using cloud storage solutions like AWS or Google Cloud Storage for media assets will improve scalability and ensure fast data retrieval.
4. **Memory Management**: Efficient memory management is crucial for handling multiple game instances. Linux’s memory management features, such as virtual memory and swap space, help optimize performance. Caching mechanisms, such as Redis, can reduce database queries and improve response times.
5. **Distributed Systems and Networks**: To support cross platform communication, the application should be designed as a distributed system using RESTful APIs. Load balancing and cloud-based networking will ensure high availability. WebSocket or similar real time communication protocols can be used to provide instant updates to players.
6. **Security**: User authentication and data protection should be enforced using encryption (SSL/TLS) for data transmissions. The system should follow security best practices such as role based access control (RBAC), secure API endpoints, and regular security patches to prevent vulnerabilities.