

# Draw It or Lose It

# **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 | 06/09/2024 | Jael Andre | Develops a detailed plan for growing 'Draw It or Lose It. |

**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)

In the rapidly expanding field of online gaming, the importance of efficiently managing game sessions, player interactions, and team dynamics cannot be overstated. Our current system struggles with maintaining unique identities and ensuring data integrity across games, players, and teams. These challenges highlight the need for a unified, scalable system that not only prevents duplications in game or team names but also supports effective tracking of player activities and maintains high performance as user engagement increases.

We propose the development of a centralized Game Management System using Java, tailored to meet the specific demands of a multi-user gaming environment. This system will incorporate sophisticated design patterns to enhance functionality and efficiency:

1. Singleton Pattern: Ensures that only one instance of the game management service exists at any given time, preventing inconsistencies and conflicts in game state management.
2. Iterator Pattern: Allows efficient navigation through collections of games, players, and teams, optimizing operations such as searches by name or ID.
3. Entity Inheritance Hierarchy: Promotes code reuse and simplifies maintenance by having all game-related entities (games, players, teams) inherited from a base Entity class, centralizing common attributes like ID and name.

**Key Features of the Game Management System:**

* Unique Identification: Implements a system that automatically assigns and tracks unique identifiers for games, players, and teams, preventing duplication.
* Efficient Search and Retrieval: Uses the iterator pattern to facilitate efficient searching by attributes such as name or ID, without exposing internal structures.
* Consistency and Integrity: The singleton pattern ensures all operations on games are centrally managed and consistent across different parts of the application.

**Benefits to the Client:**

* Reduced Complexity: Centralizing the management of games significantly simplifies handling game states across multiple platforms and interactions.
* Scalability: The proposed design inherently supports scaling, capable of managing an increasing number of users and games without a substantial increase in overhead.
* Reliability: Ensures high reliability and integrity of gaming data, which is crucial for maintaining user trust and engagement.

**Conclusion and Next Steps:**

The design patterns and system architecture we propose are specifically designed to meet the robust demands of a multi-user game management system. By moving forward with this proposal, the client will benefit from a streamlined, efficient, and scalable system capable of supporting a large and active user base. The next steps will involve detailed system design and development phases, followed by iterative testing and deployment cycles to ensure the system meets all operational requirements. By embracing this solution, the client can ensure that their gaming platform remains competitive, responsive, and capable of delivering superior user experience.

## 

## Requirements

To ensure the proposed Game Management System aligns with the client's expectations and operational demands, we've meticulously defined the necessary business and technical requirements. Each game, player, and team will be assigned a unique identifier to avoid conflicts, ensuring the gaming environment remains organized. The integrity and consistency of data across all gaming sessions and interactions are crucial for maintaining user trust and fairness. The system is designed to scale smoothly as the user base expands, handling increased loads without sacrificing performance. Operational efficiencies will be achieved by automating routine management tasks, therefore freeing up resources for strategic initiatives. From a technical standpoint, we will implement a singleton pattern to centralize session management and use an iterator pattern for efficient data access and modifications. An entity base class will foster code reusability and simplify maintenance. The system will seamlessly integrate with existing infrastructure, and robust security measures will be implemented to protect user data and prevent unauthorized access. By meeting these requirements, the Game Management System will not only satisfy current needs but also provide a foundation for future growth, ensuring alignment with the client’s strategic goals for successful implementation and operation.

## [Design Constraints](#_2et92p0)

Designing a game management system for a web-based distributed environment requires addressing several key constraints that directly impact performance and user experience. A major challenge we face is network latency and limited bandwidth; we mitigate these issues by implementing efficient communication protocols and optimizing data handling to keep the game responsive. As our user base expands, our system is designed for smooth scaling, utilizing techniques like load balancing and distributed caching. We also manage the complexities of multiple users accessing or modifying data simultaneously by incorporating strong concurrency control mechanisms and transaction management to ensure data integrity and consistency. Security is a top priority, so we safeguard user data with encrypted transfers and thorough access controls. Our system's architecture is modular and built on microservices, enhancing maintainability and allowing for easy adaptation to evolving requirements. We also adhere to legal standards such as GDPR and COPPA, implementing strict data privacy measures and regularly updating our compliance procedures. By strategically selecting technologies and methodologies that overcome these challenges, we are dedicated to developing an efficient, scalable, and engaging game management system for a distributed setting.

## [Domain Model](#_8h2ehzxfam4o)

**Class Relationships and Structure**

1. **Entity Class**:

* This is a base class that provides common attributes such as id (long) and name (String) for other entities. It features common methods like constructors, getters for ID and name, and a toString() method for easy output. This demonstrates the OOP principle of **inheritance**, where Game, Player, and Team classes inherit shared properties and behaviors from Entity.

1. **GameService Class**:

* Acts as a singleton (noted by the private static instance variable and private constructor), ensuring only one instance of this class is created throughout the application. This is a direct implementation of the **Singleton Pattern**, optimizing resource usage and providing a global point of access.
* Contains a list of Game objects and methods to add and retrieve games, managing game entities throughout their lifecycle.

1. **Game Class**:

* Inherits from Entity and extends its functionality by including a list of Team objects. This relationship is depicted by the aggregation link, indicating that a game can have zero or many teams.
* Encapsulates game-specific attributes and behaviors, like adding a team, which adheres to the **encapsulation** principle by managing the state within the object itself.

1. **Team Class**:

* Also inherits from Entity. It contains a list of Player objects and methods to manage these players, like how Game manages Team.
* This setup uses **composition** to form a strong association between Team and Player, where a team consists of one or many players.

1. **Player Class**:

* Inherits common functionalities from Entity and focuses on player-specific attributes and methods.

**Object-Oriented Principles Demonstrated**

* **Inheritance** is used throughout the model to avoid redundancy and enhance code reusability. This is visible in the Entity class extending to Game, Player, and Team.
* **Encapsulation** is employed in each class to keep the data (fields) and the methods that modify the data together, protecting object integrity by preventing external access to sensitive data directly.
* **Polymorphism** could be implied here where methods like toString() are overridden in derived classes to provide specific implementations.
* **Abstraction** is used in the design to provide simplified, high-level interfaces for interacting with complex realities. Each class provides methods that abstract the complexities of each entity’s management.
* **Singleton Pattern** in GameService ensures that game management is handled uniquely throughout the application, providing a single point of configuration and operation, which is critical in a gaming environment where game state consistency is paramount.

This architecture efficiently fulfills software requirements by ensuring data integrity, reducing redundancy, and simplifying maintenance and scalability of the game management system.

**.**

## 

## "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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers might not be the first thing that comes to mind in web hosting, but don’t count them out. They’re incredibly stable, boast excellent security features, and fit like a glove with other Apple products. This integration can be a game-changer for certain specialized applications. | Linux is the go-to for server-side applications thanks to its reliability, flexibility, and cost-effectiveness. It supports a vast array of software and languages and is highly customizable, making it perfect for web applications. Its open-source nature means you can tweak it endlessly to fit your specific needs. | Windows servers are known for their ease of use and seamless integration with other Microsoft products, making them a solid choice for applications needing deep Windows-based infrastructure integration. However, be prepared for higher licensing costs and potentially less customization compared to Linux. | Mobile devices usually connect to remote servers rather than host server-side applications. The focus here is on ensuring smooth, lightweight client-server communication to handle mobile data limitations and variable network conditions. |
| **Client Side** | On the client side, Macs are known for their sleek performance, user-friendly interface, and strong security. Developing for Mac means you’ll need to consider the higher cost of hardware and software licenses. However, the development environment is top-notch, with Xcode leading the way, offering everything you need to create amazing applications. | While Linux might not be the first choice for the average user, it’s beloved by developers and tech enthusiasts. Developing for Linux clients means dealing with a variety of distributions and desktop environments, which can make testing more complex but also more rewarding. | Windows rules the desktop market, making it essential for any client-side application. You’ll need to consider the diverse range of devices, from high-end PCs to budget-friendly tablets, which will impact your design and testing efforts. | Developing for mobile means tackling a variety of screen sizes, operating systems, and hardware specs. iOS and Android dominate the scene, each with its own development environment (Swift/Obj-C with Xcode for iOS, Java/Kotlin with Android Studio for Android). |
| **Development Tools** | Xcode is the star player for native Mac applications, but other powerful IDEs like IntelliJ and Visual Studio Code are also in the mix for web apps. When coding for Mac, Swift and Objective-C are your main tools, while Ruby, PHP, Python, and JavaScript handle server-side tasks. | On Linux, you’ll find a range of tools like Eclipse, IntelliJ, and native editors such as Vim and Emacs. Python, Java, C, and C++ are your go-to languages for both backend services and desktop applications. | Tools: Visual Studio is a powerhouse for Windows development, supporting languages like C#, .NET, and Visual Basic. Other popular tools include JetBrains Rider and Unity, especially for game development and applications with rich graphical interfaces. | Android Studio and Xcode are essential for mobile development. Cross-platform frameworks like Flutter and React Native are invaluable, enabling you to share codebases across iOS and Android, saving time and resources while reaching a wider audience. |

## Recommendations

1. **Choosing the Ideal Platform**  
   In a world where your users might be tapping on a tablet one minute and clicking through on a laptop the next, going cross-platform is your best bet. Whether you're thinking about spinning up a web app or diving into game development with Unity, you're covering your bases well. Unity shines by letting you push your project out on everything from Windows and macOS to iOS and Android—nobody's left out. For webbies, blending HTML5, JavaScript, and CSS with backend heroes like Node.js or ASP.NET Core means smooth sailing on any device.
2. **Mastering OS and Architecture Varieties**  
   Unity's your quiet powerhouse—it quietly handles the messy details of differing operating systems so you can focus on the fun part: creating. It's ready for any hardware you throw at it, whether it's ARM for mobile or x64 for desktops. Web projects? They live in the browser, making them naturally agile across operating systems. You code, they adapt—it's as simple as that.
3. **Smart Storage Strategies**  
   With all the digital bits and bobs your app needs, consider stashing your assets in cloud storage like Amazon S3, and tuck your gameplay data and user profiles into DynamoDB or MongoDB. These are not just shelves for your data—they're smart, scalable solutions that serve up your data fast, no matter where your users are.
4. **Managing Memory Like a Pro**  
   Unity isn't just about making games look good—it helps them run smoothly by managing what's loaded into memory and when. It's like having a backstage crew in a theater, making sure the right props are on stage at the right time. For web apps, the browser's the brain, handling memory with finesse thanks to engines like V8 in Chrome and SpiderMonkey in Firefox, ensuring your app doesn't skip a beat.
5. **Networking and Distribution Dynamics**  
   For games that thrive on real-time player interaction, WebSockets are your go-to for that instant connection in web apps. And if you're using Unity, plug into Photon Networking or Unity Multiplayer Services to get that seamless multiplayer vibe across different platforms. Keep things smooth with smart network tricks like load balancing and using CDNs to speed up content delivery.
6. **Locking Down Security**  
   Security’s your top priority:
   * **Data Encryption**: Wrap your data in TLS for transit and AES when it rests.
   * **Access Controls**: Gatekeeping with OAuth and RBAC means users only get what they need, nothing more.
   * **Stay Sharp with Audits**: Regular security check-ups and sticking to rules like GDPR not only keep the bad guys out but also build trust with your players around the globe.

By weaving together these strategies, "Draw It or Lose It" will not only stand out for its universal compatibility and performance but will also provide a secure, engaging, and dynamic experience for every user, everywhere.