

The Gaming Room

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design** 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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

[**Recommendations 8**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 02/19/2025 | Fabiola Soyoy | Updated the recommendations section with detailed analysis of operating platforms, system architectures, storage management, memory management, distributed systems, and security considerations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is expanding their Android game “Draw it or Lose it” to be able to run on multiple platforms by using a web based application. The new version of the game will support multiple teams with multiple players, and make sure that the games and teams have their own distinct id/name. This will help make sure that only one instance of the game exists in memory at any time.

## Requirements

**Business Requirements**

* Support multiple teams with multiple players in a single game session.
* Make sure game and team names are unique.
* Expand Draw It or Lose It into a web based application compatible with multiple platforms.

**Technical Requirements**

* Make sure only one instance of the game exists in memory at any given time.
* Use effective design patterns to support scalability and enforce uniqueness for game, team, or player names.

## [Design Constraints](#_2et92p0)

1. **Platform Compatibility**
   * **Constraint:** The application must run flawlessly on multiple platforms.
   * **Implications:** This requires the use of cross platform web frameworks for efficient compatibility across browsers.
2. **Singleton Design Pattern**
   * **Constraint:** The application must make sure that only one instance of the game is active at a time.
   * **Implications:** Implementing the GameService class as a singleton will ensure a single instance is created, managing game memory efficiently.
3. **Name Uniqueness**
   * **Constraint:** Games, teams, and players must have unique names to avoid conflicts.
   * **Implications:** Iterative methods within the addGame and addTeam functions must validate name uniqueness, ensuring scalability as the application grows.

[**System Architecture View**](file:///C:\Users\blank\Downloads\CS%20230%20Project%20Software%20Design%20Template.docx#_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)

1. **Entity Class**

* Acts as a base class for Game, Team, and Player, sharing common attributes like id and name.
* Ensures consistency and reduces redundancy.

1. **GameService Class**

* Implements the Singleton pattern to ensure only one instance of the service exists.
* Manages the creation and retrieval of games, teams, and players.

1. **Game Class**

* Represents a single game and maintains a list of associated teams.
* Relies on the GameService class for lifecycle management.

1. **Team and Player Classes**

* Teams manage multiple players, and both inherit from the Entity base class.

**Object-Oriented Programming Principles**

* **Encapsulation**: Each class manages its own data (e.g., Game manages its teams, Team manages its players).
* **Inheritance**: Shared attributes (id, name) are centralized in the Entity class and reused by derived classes.
* **Polymorphism**: Classes override the toString() method to provide specific implementations for Game, Team, and Player.

**"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](file:///C:\\Users\\blank\\Documents\\School\\CS-230%20Operating%20Platforms\\CS%20230%20Project%201%20Software%20Design%20Soyoy.docx" \l "_2o15spng8stw)**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac offers strong stability and user experience for web hosting. However, it’s more expensive compared to Linux or Windows. It is ideal for creative or small-scale applications. | Linux is cost-effective, reliable, and highly customizable, making it popular for web hosting. It is open-source but requires skilled configuration. | Windows provides user-friendly hosting and is compatible with enterprise-level applications. However, Linux is generally more trusted for large-scale web hosting. | Mobile hosting requires flexibility and optimized frameworks for platform compatibility, but fragmentation between iOS and Android can complicate deployment. |
| **Client Side** | Development on Mac involves proprietary tools like Xcode and languages like Swift or Objective-C, which can be expensive. | Linux clients use open-source languages like Python or Java, making them cost-effective, but they require specific knowledge. | Windows development is straightforward using tools like Visual Studio, requiring C# or C++. Windows applications offer good compatibility but require licenses. | Mobile development needs responsive frameworks like Flutter to simplify app deployment across iOS and Android. Handling device fragmentation is crucial. |
| **Development Tools** | Mac tools like Xcode and Swift are powerful but limited to Apple devices, which can restrict deployment options. | Linux offers open-source tools like Eclipse and NetBeans. They are flexible but require command-line knowledge for setup. | Windows uses Visual Studio and .NET, which provide robust functionality. However, licensing costs and resource demands can increase development costs. | Mobile development requires cross-platform tools like Flutter or React Native to handle development efficiently for both Android and iOS platforms. |

## Recommendations

**Operating Platform**

For Draw It or Lose It a Linux based server platform like Ubuntu or CentOS is recommended due to their scalability, security, and cost effectiveness. Linux servers provide a stable and open source environment, reducing licensing costs. Additionally, it supports containerization for better deployment and management of the game’s distributed architecture.

**Operating System Architecture**

The Linux operating system follows a monolithic kernel architecture. This means that it manages system calls, memory, and I/O operations within a single unified system. This architecture is highly efficient and stable which makes it suitable for hosting web applications with multiple concurrent users. Additionally, it provides support for virtualization using KVM, Docker, and VMware, enabling scalable implementation.

**Storage Management**

Given the large image library, cloud based storage solutions such as Amazon S3, Google Cloud Storage, or Microsoft Azure Blob Storage would be ideal. These platforms offer automatic scaling, redundancy, and security features. Additionally, a NoSQL database like MongoDB or a relational database such as MySQL/PostgreSQL can be used to store user profiles, game progress, and leaderboard data efficiently.

**Memory Management**

The recommended Linux server uses paging and swapping for efficient memory management. It allocates memory dynamically, ensuring efficient handling of multiple game instances. Memory caching techniques like Redis or Memcached can be integrated to reduce load times and improve performance. The server also supports garbage collection for efficient memory cleanup which ensures a smooth operation even with multiple users.

**Distributed Systems and Networks**

To support multi-platform communication, a microservices architecture utilizing RESTful APIs or WebSockets is recommended. This setup allows different platforms to interact with the game efficiently. High availability and stability can be ensured by distributing incoming traffic evenly within several servers with a load balancer like Nginx. Also using a content delivery network like Cloudflare will help reduce latency enhancing performance for users in different locations by caching and delivering content more efficiently.

**Security**

Priority one when managing user data across platforms is security. To protect user information the following security measures will be implemented:

* Data encryption: all sensitive user data will be encrypted using AES 256.
* Secure authentication: users will authenticate using OAuth 2.0 or multi factor authentication.
* Firewall and DDoS protection: a firewall and intrusion detection system will be used to monitor threats.
* Regular updates & security patches: the system will follow best security practices to ensure protection from vulnerabilities.