

Draw It or Lose it

**CS 230 Project Software Design**

Version 1.0

**Table of Contents**

**Document Revision History**

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/20/2024 | Jacob Hasbrook | Initial creation of the game application design. |
| 1.1 | 10/04/2024 | Jacob Hasbrook | Updates to Evaluation and Recommendations sections. |
| 1.2 | 10/17/2024 | Jacob Hasbrook | Redesigned and further defined recommendation’s section. |

**Executive Summary**

Client: The Gaming Room

In this project, the client needs the code base for Draw It or Lose It to be developed. Draw It or Lose It is a web-based game application, which is designed to expand from the Android only platform. The project aims to be a scalable, web-based version of the game that can support multiple teams, and players with unique identifiers. The games consist of one active instance running and anytime, using a Singleton pattern. The solution will provide us with an application that is cross-platform allowing it to function on devices across the board.

**Requirements**

*The client has requested the following features:*

The game application, must be scalable and capable of handling simultaneous client-server connections.

The game must be able to manage multiple games, teams, and players with unique identifiers and names.

The system must ensure that there is only one instance of the game management service.

The design should support future expansion to multiple platforms (e.g., desktop and mobile).

**Design Constraints**

The system must support multiple users across various platforms, meaning that the server should handle concurrent requests efficiently without data conflicts.

Ensuring that all game, team, and player names are unique requires consistent state management across the distributed environment.

The game must handle user data securely across platforms, requiring robust authentication and encryption mechanisms.

The game needs to run on multiple operating systems and web browsers, which requires mindfulness of cross-platform libraries and frameworks.

**System Architecture View**

The game application of Draw It or Lose It will be designed in a client-server architecture, where the server will conduct all gaming businesses, supervise players and tames, and ensure no two games or teams share names. This will also see it store information about players and games in one central repository. On the other hand, the client part will act as the view for users to interact with the game through different devices such as desktops, mobile phones or browsers.

**Domain Model**

*Entity Class*: The base class for all game-related objects is the Entity class. This class has two common attributes which are id and name. It simplifies the code by ensuring that all game, player and team entities have a unique identifier along with a name. This shows good programming practices on the part of inheritance, based on DRY principle because it helps collect common attributes in one place.

*Game*: The game is the sub class of the Entity class, which implies that an individual game possesses an exclusive identification and title. Likewise, a list of Team objects, which are the teams that are participating in a game, are held within the Game class. The multiplicity representation (0..\*) signifies that a game is able to have none or a number of teams.

*Team*: The Team class inherits from Entity like Game class. It has a unique id and name for each team. A team may have multiple players or none hence indicated in the diagram as (0..\*) of multiplicity.

*Player*: The Player class also inherits from the Entity class such that every player is given an identifying name and number. A player belong to a team, which is represented as a one-to-many relationship (0..\*). A player comes together with one team but may play for many games being part of the Game class.

*GameService*: The purpose of the GameService class is to keep a record of all games on the system; it comprises: A collection of Game objects that will be used for taking care of more than a single one.

It has static properties which are used to monitor changes made by different entities within the system as far as games are concerned which includes next available IDs such as nextGameId, nextPlayerId, nextTeamId. The GameService has methods that will help it store new games (addGame), attach players to teams on a particular game (addTeam), as well as get game details (getGame).

It is worth noting that in this case we have implemented GameService as Singleton ensuring that there can be only one object instance created using GameService at any time thus centralizing control over games, teams and players.

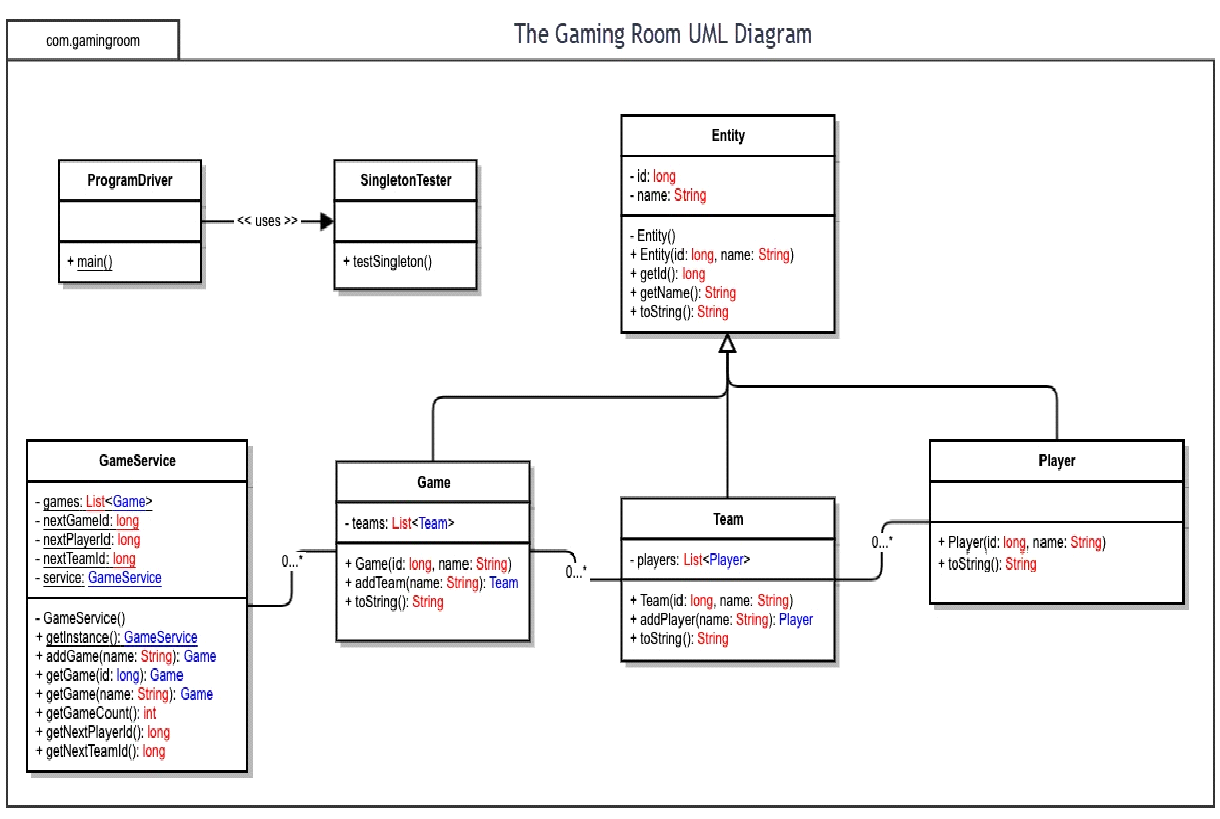
*ProgramDriver and SingletonTester*:

These classes are the primary entry points for the application while also serving as utility testers.In order to test and then initialize the game, ProgramDriver has a main method.

SingletonTester’s purpose is to ascertain if it works fine in terms of single instance by testing GameService singleton.

*Object-Oriented Programming Principles Demonstrated*:

The Entity class is used to demonstrate inheritance and serves as a base class for Game, Team and Player classes in order to avoid duplication and have common properties in one place. Composition is used in both Game and Team classes so that they can store collections of team and player objects correspondingly. This indicates the whole-part relationship between the game and its teams, or between teams and their players. The GameService class employs Singleton pattern which allows only one instance of the game service to exist therefore making centralization of game management possible. While adding new entities or making name unique, the GameService class employs loops that iterate through the collections of games, teams and players.



**Evaluation**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Because of its Unix-based operating system, Mac provides a dependable and robust server hosting environment. Its substantial expense and restricted hardware adaptability render it less appealing for extensive implementation.Mac offers stability and a Unix-based environment, making it ideal for server hosting. However, it’s more expensive compared to Linux. | The most adaptable and widely used platform for hosting web applications is undoubtedly Linux. In addition to being cost-effective, it’s open-source design, scalability, and security features make it perfect for high-traffic situations. | Windows interacts nicely with Microsoft services and offers strong support for server applications at the enterprise level. But license fees are much more expensive, particularly when scaling to thousands of users. | Generally speaking, mobile devices don't act as hosts for servers. Server-side hosting is irrelevant for this platform since they interact with server-hosted programs as clients instead. |
| **Client Side** | Swift and Objective-C knowledge, as well as Apple's Xcode, are prerequisites for developing for macOS and iOS. Although it guarantees compatibility with Apple products, there are drawbacks, such as the expensive hardware and stringent app store restrictions. | Although Linux is less ubiquitous as an operating system for consumers, developers frequently use it. Development for Linux clients may necessitate specific knowledge because of its diverse user base and distributions. | Since Windows is still the most popular desktop operating system, support for it is imperative. However, more development time and resources might be needed for testing across Windows versions. | For a wide audience, mobile platforms (Android and iOS) are essential. Having proficiency with Kotlin/Java and Swift is necessary while developing mobile applications. Cross-platform frameworks such as React Native or Flutter can streamline the procedure and cut expenses. |
| **Development Tools** | The default IDE for iOS and macOS development is called Xcode, and it needs macOS hardware to function. Primary languages are Objective-C and Swift. Although strong, the reliance on hardware raises the overall cost of development. | Numerous open-source technologies, such as Eclipse, IntelliJ, and Visual Studio Code, are beneficial to Linux development. Languages like Python, Java, and C++ are in demand. Although the platform's open-source nature lowers license costs, its diverse distributions might necessitate additional work. | Visual Studio is the most popular integrated programming environment for Windows, supporting C#, C++, and Python. Nevertheless, Visual Studio licensing may increase expenses, particularly for larger teams. | Android Studio (for Java/Kotlin) and Xcode (for Swift) are needed for mobile development. As an alternative, the development process can be streamlined and made more effective by using cross-platform frameworks like React Native and Flutter, which eliminate the need for separate codebases. |

**Recommendations**

### *Operating Platform:*

* **Requirement**: The platform must support the expansion of *Draw It or Lose It* to various computing environments, providing a stable and scalable server environment for a web-based game.
* **Recommendation**: Linux is the operating system best suited for this project. Alternatives like Windows Server or macOS can take over web services; Linux stands out for its affordability, scalability, and flexibility. Linux is open-source which allows for deep customization and customization to meet the specific needs of the game. It also provides extensive support for containerization technologies like Docker and orchestration tools like Kubernetes, which are essential for Draw It or Lose It scaling across environments. Its community driven updates with documentation make it better for the long term projects like this game.

### *Operating Systems Architectures:*

* **Requirement**: The architecture must efficiently handle multiple concurrent users, offer strong resource management, and enable high performance for the game.
* **Recommendation**: Linux’s multi-user multi-tasking architecture is ideal for optimizing resources in situations where thousands of users may be playing simultaneously. The monolithic kernel design ensures efficient use of system calls, task scheduling, and process management. This design helps build a high performing robust platform that can handle multiple concurrent connections without sacrificing speed along with Linux's support for multiple processors and hardware speeds for performance.

### *Storage Management:*

* **Requirement**: The storage system must efficiently manage game data such as player profiles, teams, and game states, and must be scalable to handle a large number of users.
* **Recommendation**: Relational databases like MySQL is great for storing structured data like game stats, player profiles, and teams in Draw It or Lose It. These databases provide robust data integrity, communication support, and scalability. The database is able to handle increased demand and distribute the load across multiple servers, ensuring the game runs smoothly even during peak usage In cloud deployments AWS offers managed database solutions with automatic backup, recovery and scaling for services.

### *Memory Management:*

* **Requirement**: The platform must handle memory management efficiently, ensuring smooth gameplay even with many concurrent users.
* **Recommendation**: Linux excels at memory management, offering advanced features such as paging and support for Huge Pages and THP.. These enable large blocks of memory to be allocated more efficiently, reducing the overhead created in smaller allocations. This is especially useful for Draw It or Lose It, where players need quick access to stats, game state and graphical assets for a smooth game experience. Memory caching tools such as memcached can be used to store frequently accessed data, and also reduce latency. Linux’s memory management tools, such as Valgrind, help detect memory leaks, making the gaming platform more stable over time.

### *Distributed Systems and Networks:*

* **Requirement**: The game must communicate between multiple platforms, ensuring that users on different devices can play together, and the system must remain available during outages or high traffic.
* **Recommendation**: To facilitate communication across platforms, web services will use the HTTP/HTTPS protocol, ensuring secure data transfer between clients and servers. Load balancers like NGINX or HAProxy will distribute incoming traffic, monitoring system availability even during periods of high traffic or partial downtime. A microservice architecture is highly recommended for Draw It or Lose It, as it breaks down the game into smaller independent tasks (e.g., user login, game state management, and image rendering) This architecture enables acceptable mistakes to be made great, because they can be scaled, updated in individual services , or replaced without impacting the rest of the game. Messaging groups such as Kafka or Redis can be used to facilitate real-time synchronization between distributed systems, ensuring consistent play across platforms.

### *Security:*

* **Requirement**: The platform must protect user data and ensure secure communication between various platforms, preventing unauthorized access and data breaches.
* **Recommendation**: Safety is one of the most important concerns of Draw It or Lose It. To protect user data during transmission, the platform will use HTTPS with TLS encryption, ensuring sensitive information is protected. For user authentication, industry-standard protocols such as OAuth 2.0 can be used to securely manage login sessions across platforms. Strong encryption algorithms such as AES-256 will be used to encrypt sensitive data such as passwords or payment information. Network isolation and firewall rules will be used to prevent unauthorized access to the system, while an intrusion detection system will monitor any malicious activity and rate- limiting and DDoS protection measures will be maintained in place to ensure the platform remains viable.