

Draw It or Lose It

# CS 230 Project Software Design

Version 1.3

## 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 3](#_Toc115077324)

Evaluation 4-7

Recommendations 8

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.3 | 10/16/2025 | Haseeb Rehman | Added characteristics, advantages and weaknesses of various platforms for client. Added Additional Recommendations for customer. |

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, a client of Creative Technology Solutions (CTS), is looking to expand their Android-based game, Draw It or Lose It, into a web-based platform that supports multiple operating environments. The application requires a scalable and secure design to manage players, teams, and games while maintaining uniqueness in naming and ensuring that only one game instance can run in memory at a time.

## Requirements

Business Requirements

* Expand the existing game from Android-only to a multi-platform, web-based solution.
* Ensure scalability for multiple concurrent teams and players.
* Guarantee unique names for games and teams.
* Provide secure access and protect user information.

Technical Requirements

* Implement Entity base class with id and name.
* Ensure single instance of GameService (Singleton).
* Use iterators to enforce uniqueness in addGame(), addTeam(), and addPlayer().
* Web-based distributed environment to support cross-platform communication.

## [Design Constraints](#_2et92p0)

* Distributed Environment: The system must support multiple clients accessing the same game remotely. This means concurrency and synchronization are required.
* Uniqueness: Game, team, and player names must be unique. Without enforcing this, conflicts or duplicate sessions could occur.
* Single Instance: Only one instance of the game can exist in memory, restricting the design to a Singleton model.
* Cross-Platform Compatibility: The system must run on Windows, Mac, Linux, and mobile browsers, limiting design to widely supported technologies (Java, web-based tools).
* Security Requirements: Sensitive player data must be protected through encryption and secure communication protocols.

## [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)

The UML diagram illustrates the class structure for the Draw It or Lose It game. At the top of the hierarchy is the Entity class, which serves as the parent class for the other core objects in the system. It defines two shared attributes, id and name, which are automatically inherited by its subclasses. This means that all major objects in the program, such as Game, Team, and Player, will have these identifying features. The diagram also shows that a Game can contain several Teams, and each Team is made up of multiple Players.

The GameService class is designed as a singleton, ensuring that only one instance of this class exists throughout the program. It manages the lifecycle of Game objects and maintains a collection of them. Through composition, GameService directly controls how games are created and accessed. Similarly, a Game manages its teams, and a Team manages its players, creating a layered structure of responsibility.

The ProgramDriver class contains the main() method, which acts as the starting point for the application. It relies on GameService to add new games, teams, and players. In addition, the diagram shows a <<uses>> relationship from ProgramDriver to SingletonTester, which is responsible for confirming that the singleton behavior of GameService works as intended.

Several object-oriented design principles are reflected in this diagram. Inheritance is shown through the relationship between Entity and its subclasses, which reuse the base attributes and methods. Encapsulation is demonstrated by how classes such as GameService keep their data private while offering controlled access through methods. Abstraction is evident in the way each class exposes only the essential operations (for example, managing games, teams, or players) without revealing how those operations are implemented internally. Polymorphism is also implied, since Game, Team, and Player can all be treated as Entity objects when needed. Lastly, the Singleton design pattern is implemented in GameService, ensuring that game management is handled consistently from a single shared instance.

"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 | macOS is Unix-based, offering excellent security, reliability, and developer support for web services. It supports server technologies such as Apache, Nginx, Node.js, and Java. However, Apple no longer supports macOS Server, and Mac hardware is expensive and not designed for large-scale server deployments. It is suitable for small development servers or testing environments but not for high-volume production. Licensing: macOS comes bundled with Apple hardware, so there are no direct OS licensing fees, but hardware costs are high. | Linux is the preferred server platform for modern distributed applications due to its scalability, performance, and cost efficiency. It supports web frameworks such as Java (Dropwizard, Spring Boot), Node.js, Python, and containerization tools like Docker and Kubernetes, making it ideal for Draw It or Lose It’s web-based architecture. It offers strong community support, high availability, and enterprise-grade security. Licensing: Free and open source, with optional enterprise support costs. | Windows Server provides robust tools and integrations such as IIS, .NET, and support for Java, Node.js, and containerized deployments. It scales well in enterprise environments and integrates easily with Active Directory and Microsoft cloud services. However, it requires frequent updates and has a larger security footprint than Linux. Licensing: Windows Server is a paid OS with per-core and Client Access License (CAL) costs. | Mobile operating systems are not designed for hosting server applications. Their purpose is to connect to the centralized web server. While they support local testing via emulators or lightweight servers, all production hosting should occur on Linux or Windows servers. Licensing: N/A for server-side hosting. |
| Client Side | macOS offers an intuitive and polished interface with strong browser support (Safari, Chrome, Firefox). Developing for Mac ensures smooth integration and high-performance graphics. The web client must be responsive and compatible with Safari’s specific rendering behavior. While development quality is high, it can increase testing costs because of Apple-specific hardware requirements. Expertise: Front-end developers familiar with macOS and Safari testing. | Linux clients rely on open-source browsers such as Chrome and Firefox. This platform provides flexibility, security, and cost efficiency with no licensing fees. Developers should test multiple distributions to ensure interface consistency and responsive design. Linux users benefit from strong performance and customization but may face minor rendering variations across desktop environments. Expertise: Front-end engineers experienced with open-source tools and browser testing. | Windows clients will make up the largest user base for the web version of Draw It or Lose It. The web app should support major browsers (Edge, Chrome, Firefox) and provide accessibility features such as keyboard navigation and high-contrast themes. Development is straightforward since most browsers on Windows fully support HTML5, CSS3, and JavaScript. Cost: No extra licensing for end users; QA testing time may be higher due to device diversity. | Mobile devices (Android and iOS) require a responsive, mobile-first design to ensure consistent gameplay across different screen sizes and orientations. The app will run in mobile browsers using HTML5, CSS3, and JavaScript. Progressive Web App (PWA) features can enable offline play, notifications, and near-native experiences. Developers must account for touch gestures, limited bandwidth, and battery efficiency. Expertise: Front-end developers with mobile-web testing experience. |
| Development Tools | macOS supports a wide variety of development environments, including VS Code, IntelliJ IDEA, and Xcode (essential for iOS testing). Common languages include JavaScript, TypeScript, Java, and Swift. The system integrates well with Git and modern CI/CD pipelines. Licensing: Most IDEs are free; Xcode is free but requires Apple hardware. | Linux offers one of the richest development ecosystems. It supports tools such as VS Code, Eclipse, IntelliJ, Docker, Kubernetes, Git, and Jenkins. It is the best environment for building and deploying backend services in a distributed architecture. Licensing: Free/open source, minimizing cost and increasing flexibility. | Windows provides strong enterprise tools, including Visual Studio, .NET, and cross-platform languages such as JavaScript, C#, and Java. Visual Studio offers powerful debugging, profiling, and API testing features. Windows Subsystem for Linux (WSL) allows running Linux containers, enhancing flexibility. Licensing: Visual Studio Community is free; Professional and Enterprise versions require paid licenses. | Mobile development uses Kotlin (Android), Swift (iOS), and JavaScript for web components. IDEs include Android Studio, Xcode, and VS Code for cross-platform frameworks like Flutter or React Native. Licensing: Mostly free tools; Apple Developer Program ($99/year) required for publishing native iOS apps. |

## 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: To expand Draw It or Lose It beyond Android, the best option is to use a web-based platform hosted on a Linux server. Linux is reliable, fast, and inexpensive compared to other options like Windows Server. It works well with modern web tools and can easily grow as more people play the game. Using a web platform means players can access the game from almost any device computers, tablets, or phones just by opening a browser, without downloading anything.
2. Operating Systems Architectures: A client–server setup is the best fit for this game. The client is what players see and interact with (the website), and the server is where the main work happens keeping track of players, teams, and scores. The server handles data and sends updates back to the players’ screens. This setup makes it easy to manage updates and keeps the game running smoothly even when many people are playing at once.
3. Storage Management: The game will need to store different types of information. A database will hold details like usernames, passwords, and game results, while a cloud storage service will store larger files such as the images used during gameplay. Using both helps keep the system organized and fast. The database manages important records securely, and the cloud storage ensures that images and files can load quickly for players anywhere.
4. Memory Management: Memory management is how a computer decides what data to keep active while the game is running. On the player’s side, web browsers already take care of this automatically they clear out memory that’s no longer needed. On the server side, the system uses built-in tools to free up unused memory so that the game runs efficiently, even when lots of players are online. This prevents crashes and helps the game load faster.
5. Distributed Systems and Networks: Because players will join from many different devices, the game will rely on a central server in the cloud that connects everyone together. The server sends updates in real time, so everyone sees the same thing at the same moment. If one connection slows down or temporarily drops, the system will retry the action automatically instead of making players restart the game. This setup also makes it easy to add more servers in the future as the player base grows.
6. Security: Keeping player information safe is extremely important. All communication between the player’s browser and the server will be protected using secure connections (HTTPS). Players will log in with passwords that are stored in an encrypted format, meaning they can’t be read even if someone gained access to the database. The system will also check that each user has permission to access certain features, preventing unauthorized actions. These security measures will protect user data and keep the game trustworthy for everyone.