

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 4**](#_Toc115077324)

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

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/30/2025 | Jasmine Garcia | First draft. |
| 2.0 | 04/12/2025 | Jasmine Garcia | Second draft. |
| 3.0 | 04/27/2025 | Jasmine Garcia | Third draft (final). |

**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 requests that CTS design a web-based version of their current Android-based game: *Draw It or Lose It*. In this popular game, multiple teams are racing against a time to guess the puzzle before time runs out.

The software can juggle multiple teams with distinct team names and games, all while ensuring that only one instance of the game exists in its memory at a time, so it doesn’t mix them up.

## Requirements

*Business Requirements:*This game allows multiple teams, with multiple players per team. Also, the game, team, and player names must be unique each time. Lastly, there’s only one game instance that exists at a time. That way, players can be sure there are no issues with duplicating in the software’s memory.

*Technical Requirements:*

The game is currently Android-based, but we must ensure it is translated to a web-based game, handling multiple platforms. Also, it must rely on Java, with web technologies for deployment. Lastly, it must use a Singleton and Iterator design pattern in order to meet all requirements mentioned above.

## [Design Constraints](#_2et92p0)

The Gaming Room already has an Android-based deployment of *Draw It or Lose It.* Therefore, the game must now be developed to work across multiple platforms (mobile and web), so that users can choose to play from different devices if they wish. Java is the best choice for this project, as it is notably compatible with both applications. To ensure the game’s memory is regularly up to date across all platforms, the game will use a Singleton pattern in the GameService class when building the code. This ensures that only one of the game’s instances is loaded in its memory at a time, which is helpful in making sure each game is distinct and reduces memory usage.

## [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 application consists of three main classes: games, teams, and players. The game class is representing a game instance, in which it contains multiple teams. Likewise, each team holds multiple players. All three classes inherit from the base class: entity. The entity offers better organization by providing id and name, which are common attributes that are shared amongst all. This helps avoid repetition by allowing shared functionality to be defined in one entity class, while still allowing the game, team, and player classes to have their own unique behaviors.

The UML for the proposed design is shown below:

**"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 is decent for small-scale development, having flexible commands for server configuration; It has built-in tools like Apache for web hosting. However, but it’s not a common server used for production. Scaling large applications might be limited as well. If the client already owns the Mac hardware, there aren’t any additional licensing costs to use this. | Linux is stable, cost-effective, secure, scalable, and has multiple web hosting options. For these reasons, it is the best choice for server-side deployment. Linus supports major web hosting software, like Apache and Nginx. Most Linux distributions are open-source and also free. | Windows can run on more software when compared to other systems, like Linux. Its servers support web hosting through internet information services, or IIS. It’s not as popular in open-source development environments. And although it has enterprise support and software compatibility, the cost of licensing can be substantial. | Mobile devices are not great for hosting game servers strictly because of their resource constraints. They have a limiting battery life, processing power, and OS restrictions. Mobile devices aren’t used for hosting the backend. However, they are highly recommended to use for the client-side use. |
| **Client Side** | Mac is great for both mobile and web-based clients that uses HTML and JavaScript. It has a similar cost to Windows-based developments as well. Mac supports more modern browsers like Safari, also tools like Xcode and Eclipse make it easy to build responsive designs. | Linux is low cost and can be used for developing web-based clients. The time expertise would depend on the client’s experience with the software. However, it’s more commonly used as a server platform, so setting up client-side testing environments might need extra configurations or browser setups. | Windows is most ideal for the client-side of development, as it has the lowest time and expertise requirements needed. It is great for HTML, CSS, and JavaScript, and supports all major browsers and development tools out of the box. | This is the lowest cost platform in terms of device accessibility, because most users already have smartphones. The time would be greater than Windows development, but it does provide a decent interactive experience for the players. Developers need to make sure the app is mobile-friendly and performs well on small screens, which might mean there’s more testing involved. |
| **Development Tools** | Swift, Eclipse, Xcode, and IDEA are all apps that Mac supports for mobile app development. iOS development requires Xcode, which is free. However, publishing apps to the App Store costs $99/year, as developers would need to enroll in the Apple Developer Program. | Atom, Eclipse, Visual Studio Code, and NetBeans are all commonly used for Java development in Linux. These are all free, open-source tools, meaning that Linux is highly cost-effective! However, there may be more technical skills required to set up the development environments within Linux. | Visual Studio, Eclipse, and Android Studio are common tools used for Windows. There are free versions of Visual Studio, but professional editing does require a fee. This platform is great for any teams that prefer enterprise integration or GUI tools. | Xcode and Android Studio are the required tools for building mobile apps. A Mac system is required and publishing the apps costs $99/year. Android only has a one-time fee of $25 for publishing through Google Play. |

## 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**: When considering an operating platform for The Gaming Room, I would recommend Mac or Windows for the client-side, and Linux for the server. In choosing Mac or Windows, this can totally be based upon which platform the client is most comfortable with. These are budget-friendly and user-friendly options that offer broad accessibility and can handle high traffic.
2. **Operating Systems Architectures**: Regarding operating systems architecture, the server should use a Linux-based monolithic Kernal architecture, which combines services like file handling, memory management, and networking into one core. The client side should include platforms such as Windows and/or MacOS, which use layered architectures that separate hardware management, application interfaces, and system services. For example, Windows services offer easy access to system resources, display graphical user interfaces (GUIs), and use multimedia, web services, and messaging.
3. **Storage Management**: By using powerful storage management tools, Linux makes handling files, system resources, and databases both reliable and easy. For Draw It or Lose It, I would recommend using MySQL to manage game data. This could include player profiles, teams, and scores. MySQL is relatively easy to use, scalable, and offers organized storage that can expand as the game becomes more popular. Additionally, Linux supports many file systems, which makes it simple to manage storage through command-line tools as needed.
4. **Memory Management**: Linux handles memory through a virtual memory and efficient memory allocation. This helps the server run smoothly, even when there are large amounts of data, like player profiles, game images, and team scores. In Draw It or Lose It, the software uses the Singleton design pattern in the GameService class, ensuring that only one instance of the game exists in memory at a time. By doing this, the memory usage is kept low and the game runs securely, mitigating conflicts and duplicates. Additionally, developers can use built-in libraries in the server environment to support smooth gameplay and manage other assets, like sound effects, animations, or background music.
5. **Distributed Systems and Networks**: In order to support cross-platform functionality, Draw It or Lose It’s server should allow for multiple platforms, including Windows, Mac, and mobile devices to connect to a single instance in the game. Using WebSockets and RESTful API, real-time communication is made reliable and quick. Hosting the game in a cloud-based environment will solidify the players’ connection from anywhere. The server failover systems will help maintain the game’s ability to run, even if there are network issues.
6. **Security**: Linux has strong, established security features built into the program. These include access controls and firewalls, which help maintain protection around user data. To further protect the game, security measures like regular security updates, secure password storage, and multifactor authentication (MFA). These kinds of protections will apply across all platforms, ensuring that the player’s information remains safe regardless of which device they choose to play from.