

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 | 05/23/23 | Jarrale Butts | In this revision, I will provide a clear and concise summary of the client’s problem with a proposed solution, including business/technical requirements, design constraints and describing the UML class diagram provided. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based game catering to multiple platforms, based on their existing Android app game, Draw It or Lose It. The solution to the multiplatform problem would be to enable the management of a multiplayer game with multiple teams and players. The important requirements for the software include: support for multiple teams with multiple players, unique game and team names, and only one instance of the game can exist in memory at any given time.

## Requirements

Business Requirements:

1. Develop a web-based game: The Gaming Room wants to expand its existing Android app game, Draw It or Lose It, to a web-based platform that can be accessed across multiple platforms.
2. Multiplayer capability: A game will have the ability to have one or more teams involved, with the ability to assign multiple players to each team.
3. Unique game and team names: The game should allow for the uniqueness of game and team names and the ability to check whether a name is in use when choosing a team name.
4. Single game instance: Only one instance of the game should exist in memory at a time to optimize system resources.

Technical Requirements:

1. Cross-platform compatibility: The web-based game should be compatible with various platforms, including desktop computers, laptops, tablets, and mobile devices.
2. Unique identifiers: The software should generate unique identifiers for each game, team, and player instance.
3. User interface: A user interface should be provided to allow players to check name availability.
4. Scalability: The software should be designed to handle both old Android and new multi-platform players.

## [Design Constraints](#_2et92p0)

Design Constraint 1: Network Delays

The time it takes for data to travel between players and the game server can cause delays in gameplay interactions. This constraint requires minimizing the impact of network delays to ensure smooth gameplay.

Implication: The game needs to be designed to so that it reduces the effects of network delays. This involves implementing techniques to make the game feel responsive, even when there is a delay in data transmission. It also means optimizing how the game communicates with the server.

Design Constraint 2: Handling Multiple Players

The game will need to be able to handle a large volume of players playing at the same time. It needs to accommodate the growing player base without slowing down or becoming unresponsive.

Implication: The game needs to consider scalability, which needs to be designed to handle increased player traffic. This may involve using multiple servers and ensuring that the game's performance remains consistent even with a large number of players.

Design Constraint 3: Security

Protecting the players data and preventing unauthorized access and cheating is an important aspect of any multiplatform game, especially when online gameplay is necessary.

Implication: Measures should be taken to ensure security of player data such as encrypting sensitive information and using secure communication protocols. The game should take measures to verify the identity of players and prevent cheating or hacking attempts.

Design Constraint 4: Multi-Browser Compatibility

The game needs to work efficiently on different web browsers which ensures a consistent experience across various platforms.

Implication: The game should consider the differences between web browsers and pay particular attention to popular browsers. This may involve adapting the game's code and design to work with different rendering engines and browser capabilities.

Design Constraint 5: Multi-Device Compatibility

The game should be able to adjust to different devices with varying screen sizes and input methods, providing an enjoyable experience regardless of the device being used.

Implication: There should be a focus on making the game responsive, meaning it can adapt its layout and controls based on the device's screen size. The game should also support different input methods, such as touch controls for mobile devices and keyboard/mouse for desktops, to accommodate various devices.

## [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 provided UML class diagram represents the Domain Model of the game within the software design template. It illustrates the structure and relationships between various classes involved in the game. Here is an overview of the classes and their relationships:

1. ProgramDriver Class:
   * Represents the main entry point of the program.
   * Contains a main() method, which serves as the starting point for the application.
2. SingletonTester Class:
   * Represents a class responsible for testing the singleton pattern.
   * Contains a testSingleton() method.
3. Entity Class:
   * Represents a base class that holds common attributes and behaviors.
   * Includes id (long) and name (String) attributes.
   * Provides constructors, getters, and a toString() method to interact with entity objects.
4. GameService Class:
   * Represents a class that manages the game-related operations.
   * Contains a list of games, along with nextGameId, nextPlayerId, and nextTeamId attributes.
   * Implements the singleton pattern by providing getInstance() method.
   * Provides methods to add and retrieve games, retrieve game count, and manage player and team IDs.
5. Game Class:
   * Represents a game entity in the application.
   * Contains a list of teams participating in the game.
   * Provides methods to create game objects, add teams, and retrieve a string representation of the game.
6. Team Class:
   * Represents a team entity within a game.
   * Contains a list of players in the team.
   * Provides methods to create team objects, add players, and retrieve a string representation of the team.
7. Player Class:
   * Represents a player entity in the application.
   * Provides methods to create player objects and retrieve a string representation of the player.

Relationships between classes:

* The ProgramDriver class uses (<<uses>>) the SingletonTester class.
* The GameService class has a composition relationship with the Game class (0...\*) and vice versa. It manages the game instances and provides methods to add, retrieve, and count games.
* The Game class has a composition relationship with the Team class (0...\*). It manages the teams participating in the game and provides methods to add teams and retrieve game information.
* The Team class has a composition relationship with the Player class (0...\*). It manages the players within the team and provides methods to add players and retrieve team information.
* The Game, Team, and Player classes all have a Aggregation relationship with the Entity class denoted with a solid line and open triangle.

Object-Oriented Programming Principles Demonstrated:

1. Singleton Pattern:
   * The GameService class demonstrates the singleton pattern by providing a single point of access to the game-related operations. The getInstance() method ensures that only one instance of GameService is created and shared across the application.

**"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** | Macs provide a suitable platform for hosting web-based software applications with support for server-based deployment methods, making it possible to host websites and web-based applications on Mac computers. Macs offer a stable and user-friendly environment, compatibility with web technologies, built-in developer tools, and robust security. Additionally, Macs seamlessly integrate within the Apple ecosystem. However, it's important to note that Macs may have limited hardware options, higher costs, limited software compatibility, and a smaller community for server administration. | Linux operating systems offer a versatile and cost-effective platform for hosting web-based software applications. With a strong focus on server functionality, Linux distributions provide robust features and wide software compatibility. The cost-effectiveness of Linux, with most distributions being open-source and free, makes it an attractive choice for hosting. However, potential licensing costs may exist for enterprise-grade distributions or additional software packages. | Windows Server provides possible web-based hosting capabilities and a user-friendly interface, broad software compatibility, robust integration with Microsoft products, and extensive support. However, potential considerations include licensing costs, limited customizability, and performance comparisons with Linux. | Mobile devices provide a portable and accessible platform for accessing web-based software applications. With touch-based interfaces, sensor capabilities, app store distribution, and push notifications, mobile devices offer enhanced user experiences. However, considerations include screen size limitations, device fragmentation, and security and privacy concerns. |
| **Client Side** | When developing software for multiple clients on Mac, consider cost, time, and expertise. Ensure compatibility by testing and optimizing for web browsers and mobile devices. Test on popular Mac browsers like Safari and iOS devices. Design a responsive user interface for different screen sizes. | Supporting multiple clients on Linux requires careful software development considerations, including cost, time, and expertise. Expertise in Linux development is essential for compatibility and optimal performance across different distributions. Compatibility with web browsers can be ensured through testing on popular Linux browsers, while mobile compatibility requires testing on Android devices and mobile browsers like Chrome. Responsive UI design and adaptive layouts accommodate various screen sizes and orientations. | Developing software for multiple clients on Windows requires specific considerations. Cost and time depend on complexity and resources. Windows development expertise ensures compatibility and performance. Test on popular Windows browsers and consider Windows-based mobile devices. Design a responsive user interface for various screen sizes. | Supporting multiple clients on mobile devices requires specific considerations. Cost and time depend on complexity and resources. Expertise in mobile app development for platforms like iOS and Android is crucial. Thoroughly test on popular mobile browsers. Use responsive web design principles for screen adaptation. Native app development may be needed for optimal user experience, requiring expertise in respective programming languages and frameworks. |
| **Development Tools** | Relevant programming languages and tools include Swift, Objective-C, and C/C++. XCode is the primary IDE for Mac development. The technical requirements for Mac development may impact the development team, requiring expertise in Swift and familiarity with XCode. Licensing costs may be associated with developer memberships in the Apple Developer Program. | Programming languages such as C, C++, Python, and Java can be used. IDEs like Visual Studio Code, Eclipse, and IntelliJ IDEA are commonly used. Linux development may require additional training or involvement of Linux specialists. Most tools and programming languages used for Linux development are open source, resulting in minimal licensing costs. | Programming languages like C#, Visual Basic .NET, C++, and JavaScript are relevant. Microsoft Visual Studio is the primary IDE. The impact on the development team depends on their experience with .NET Framework or .NET Core and Windows-specific APIs. Licensing costs may be associated with different editions of Visual Studio, but free versions like Visual Studio Community are available. | iOS apps are developed using Swift or Objective-C, while Android apps use Java or Kotlin. XCode is the IDE for iOS development, and Android Studio is used for Android development. Mobile app development requires expertise in the respective programming languages and familiarity with the IDEs. Licensing costs are typically associated with developer memberships in the respective app stores, such as the Apple Developer Program or Google Play Console. |

## Recommendations

1. **Operating Platform**: Based on the requirements and evaluation, the recommended operating platform for The Gaming Room to expand Draw It or Lose It to other computing environments is a web-based solution. A web-based platform offers cross-platform compatibility, allowing users to access the game from different devices and operating systems. By leveraging web technologies, the game can be accessed through popular web browsers on desktop computers, laptops, tablets, and mobile devices, ensuring a wide reach for players.
2. **Operating Systems Architectures**: The chosen operating platform for a web-based solution would be based on a client-server architecture. The client side would consist of various web browsers running on different operating systems, such as Windows, macOS, and Linux. The server side would require a compatible server operating system, which can vary based on the organization's preferences and expertise. Common choices include Linux-based distributions like Ubuntu Server, CentOS, or Windows Server operating systems.
3. **Storage Management**: For the recommended operating platform, a suitable storage management system would be a relational database management system (RDBMS) such as MySQL, PostgreSQL, or Microsoft SQL Server. An RDBMS provides robust data storage, efficient query capabilities, and data integrity features necessary for managing game-related information, such as user profiles, game progress, and leaderboard data. The choice of the specific RDBMS would depend on factors like scalability requirements, performance considerations, and the organization's existing infrastructure.
4. **Memory Management**: The recommended operating platform, especially for web-based solutions, relies on the memory management techniques provided by the underlying web server and programming frameworks. The web server and application frameworks handle memory allocation and deallocation, ensuring optimal usage of system resources. Additionally, modern web browsers have their own memory management systems to handle JavaScript execution and resource allocation efficiently. By utilizing best practices in web development, such as minimizing memory leaks and optimizing code performance, the Draw It or Lose It software can effectively utilize memory resources across platforms.
5. **Distributed Systems and Networks**: To enable communication between various platforms, a distributed software architecture can be employed. This can involve utilizing technologies such as Representational State Transfer (REST) APIs or WebSocket for real-time communication. The game server would serve as the central component, facilitating data exchange and synchronization between different devices. The network that connects the devices should support reliable and secure communication, ensuring minimal latency and handling potential connectivity issues. Redundancy measures, such as load balancing and failover mechanisms, can be implemented to mitigate outages and ensure an uninterrupted gameplay experience.
6. **Security**: Ensuring the security of user information is crucial for The Gaming Room. The recommended operating platform should provide robust security capabilities. This includes implementing secure communication protocols (e.g., HTTPS) to encrypt data transmission between clients and the server. User authentication and authorization mechanisms should be implemented to protect user accounts and prevent unauthorized access. The recommended operating platform should have built-in security features, such as access controls, secure storage of sensitive data (e.g., passwords), and protection against common security threats like cross-site scripting (XSS) and SQL injection attacks. Regular security audits and updates should be performed to address emerging vulnerabilities and ensure ongoing protection for the game and user data.