

# 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 | 07/08/23 | Vivian Nguyen | Update Executive Summary, Requirements, Design Constraints, System Architecture View, Domain Model, Evaluation and Recommendations for project Draw It or Lose It |
| 2.0 | 07/20/23 | Vivian Nguyen | Modified Evaluation Section |
| 3.0 | 08/06/23 | Vivian Nguyen | Modified Recommendation Section |

## [Executive Summary](#_sbfa50wo7nsh)

The goal is to design a web-based gaming application called "Draw It or Lose It" that can be played online, inspired by the 1980s television game "Win, Lose or Draw." The client's requirements include the ability to have multiple teams with multiple players assigned to each team, ensuring that game and team names are unique for easy identification and preventing duplicate names. Additionally, the client wants to ensure that only one instance of the game exists in memory at any given time by assigning unique identifiers to each game, team, and player. To address these requirements, a solution involves creating a database to manage game and team information, assigning unique identifiers and names to ensure their uniqueness, and implementing session management to handle game instances. It includes features such as player assignment to teams, user authentication, real-time communication, data validation, and error handling.

## Requirements

Business Requirements:

* The game application should support multiple teams, allowing for competitive gameplay.
* Each team should be able to have multiple players assigned to it, promoting collaborative participation.
* Game and team names should be unique to avoid confusion and enable users to easily identify their chosen teams.
* Only one instance of the game should be allowed to exist in memory at any given time, ensuring a single active game session.

Technical Requirements:

* The application should implement a database system to store and manage game-related information (game, team, and player data).
* Develop optimized database queries for efficient retrieval of game and team information and implement mechanisms to verify the uniqueness of game and team names during creation and updates.
* Unique identifiers should be assigned to each game, team, and player to differentiate instances and avoid conflicts.
* The game application should be developed as a web-based solution, accessible through a browser.

## [Design Constraints](#_2et92p0)

1. The game application requires a database system to store and manage game-related information, including game instances, teams, and players. It implies that the database should efficiently handle the retrieval and verification of unique names. To achieve this, the database design needs to ensure that unique names can be quickly and accurately accessed. To protect sensitive user data stored in the database, security measures should also be implemented. Therefore, it is important to choose a suitable database management system.
2. To guarantee that only one instance of the game is active in memory at any given time, unique identifiers must be assigned to each game, team, and player. This requires implementing a robust mechanism for generating identifiers that ensure uniqueness across instances and avoid collisions. These identifiers should be efficiently stored and managed in the database, making it easy to reference and retrieve them. Additionally, special attention should be given to handle situations where multiple users attempt to create new games, teams, or players simultaneously, ensuring conflict resolution and maintaining data consistency.
3. The game application needs to be created as a web-based solution, meaning it will be accessed and run on different client machines through web browsers. It should be designed to handle multiple users simultaneously, as many people may be accessing the game from different devices at the same time. To achieve this, we will utilize web technologies like HTML, CSS, and JavaScript for the user interface. Additionally, we'll use a backend framework such as Node.js, Django, or Ruby on Rails for server-side processing to handle the game logic and communication with the client machines.
4. The game application needs to offer a user interface that is intuitive and engaging, enhancing the overall gameplay experience. This involves designing a responsive user interface that can adapt to various screen sizes and devices, ensuring a consistent experience for all users

## [System Architecture View](#_ilbxbyevv6b6)

**Logical View:**

* Game Engine: Manages game logic, rules, and mechanics.
* User Interface: Handles the presentation and interaction with the game through web-based interfaces.
* Database: Stores and retrieves game-related data, including game instances, teams, and players.
* External Services: Integrates with external services or APIs for additional functionalities like authentication and real-time communication.

**Development View:**

* Development Environment: Utilize Java and an IDE like Eclipse.
* Version Control: Implement Git for source code versioning and collaboration.
* Development Teams: Include backend developers, frontend developers, database administrators, and a quality assurance team.
* Testing and Quality Assurance: Perform unit testing, integration testing, user acceptance testing, and employ tools for continuous integration.

**Process View:**

* Request Handling: How user requests are processed and routed through the system.
* Game Session Management: Management of game sessions and player interactions.
* Data Flow: Movement of data between different components, such as from the user interface to the game engine and database.

**Physical View:**

* Servers and Hosting: Physical or virtual servers where the application is deployed.
* Networking: Communication protocols and infrastructure used for client-server interactions.
* Storage: Physical or cloud-based storage for database and game-related assets.

## [Domain Model](#_8h2ehzxfam4o)

**"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.**

The UML class above includes following classes:

1. **Entity**: This class serves as the base class for **Game**, **Team**, and **Player** classes. It contains the common attributes ID and name, along with their corresponding getters.
2. **Game**: This class represents a game and contains a list of **Team** objects. It has an overloaded constructor that takes an identifier and name. It also has a method *addTeam()* that allows adding teams to the game.
3. **Team**: This class represents a team and contains a list of **Player** objects. It also has an overloaded constructor and a method *addPlayer()* for adding players to the team.
4. **Player**: This class represents a player and has a constructor and a *toString()* method.
5. **GameService**: This class is a singleton service for the game engine. It maintains a list of active games (**Game** objects) and provides methods for adding games, retrieving games by ID or name, and getting the count of active games. It also keeps track of the next available IDs for games, teams, and players. The **GameService** class demonstrates the Singleton design pattern, ensuring that only one instance of the class exists.
6. **ProgramDriver**: This class serves as the entry point for the application. It contains the *main()* method, which is the starting point for the program execution. In the *main()* method, an instance of the **GameService** singleton is obtained using the *getInstance()* method. It demonstrates how to access the singleton instance to perform game-related operations.
7. **SingletonTester:** a class to test singleton behavior of programming.

The relationships in the diagram are as follows:

* **GameService** has an association with **Game** with a multiplicity of 0 or more (0...\*). It means that **GameService** can be associated with multiple instances of the **Game** class.
* **Game** has an association with **Team** with a multiplicity of 0 or more (0...\*). It indicates that a **Game** can have zero or more **Team** instances associated with it.
* **Team** has an association with **Player** with a multiplicity of 0 or more (0...\*). It shows that a Team can have zero or more **Player** instances associated with it.
* The arrow with a hollow triangle at the end indicates the inheritance relationship. It means that **Game**, **Team**, and **Player** are subclasses of the **Entity** class, and they inherit the attributes and behaviors defined in the **Entity** class.

Object-Oriented Programming (OOP) principles demonstrated in the above UML diagram, particularly:

1. **Inheritance**: The classes **Game**, **Player**, and **Team** inherit from the **Entity** base class. This inheritance allows the derived classes to inherit the shared attributes and behaviors defined in the **Entity** class. Specifically, the subclasses can access and use the ID and name attributes, as well as the corresponding methods to retrieve their values. This approach promotes code reuse and establishes a clear hierarchy among the classes, simplifying the development and maintenance of the program.
2. **Encapsulation**: The **Entity** classes encapsulate their attributes and expose them through getter methods (*getId*(), *getName*()), which limits their access to the subclasses (**Game**, **Team**, **Player**) and classes within the same package. This encapsulates the internal data within the class hierarchy and restricts direct access to these attributes from external classes. Encapsulation ensures data privacy and allows controlled access to the attributes. It promotes better code organization and helps maintain the integrity of the data.
3. **Polymorphism:** The *toString()* method is overridden in the Game, Team, and Player classes, which are subclasses of the Entity class. Each subclass provides its own implementation of the *toString()* method, which is invoked when the *toString()* method is called on an object of the respective subclass. By overriding the *toString()* method, the subclasses customize the string representation of the objects based on their specific attributes and behaviors.
4. **Abstraction:** The **Entity** class serves as an abstract base class, defining common attributes and behaviors for entities in the gaming system. The abstract method *addTeam()* in **Entity** enforces a contract that each subclass must implement. Abstraction allows for code reusability, modular design, and the ability to add new entity types with specific implementations.

## [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** | **Characteristics**: macOS is a Unix-based operating system developed by Apple, known for its user-friendly interface and integration with Apple hardware and software.  **Advantages**: macOS provides a solid foundation for web development, and it is widely used by developers. It offers excellent performance and security.  **Weaknesses**: Mac hardware can be expensive. Additionally, macOS is primarily designed for end-users and might not be the most efficient choice for large-scale server deployments.  **Server-Based Deployment:** While macOS is capable of hosting web applications, it is less common in large-scale server environments due to its higher cost and hardware focus.  **Licensing Costs:** macOS is not free, and the client would need to purchase licenses for the operating system, which could be a significant cost factor. | **Characteristics:** Linux is an open-source operating system known for its stability, security, and flexibility. It offers a wide range of distributions that can be tailored to specific needs.  **Advantages:** Linux is cost-effective as it is open-source and does not require licensing fees. It has strong community support, frequent updates, and is well-suited for server deployments.  **Weaknesses:** It may require more technical expertise to set up and configure compared to other platforms. Some proprietary software might not have native Linux support.  **Server-Based Deployment:** Linux is commonly used for server hosting and has various server distributions like Ubuntu Server, CentOS, and Debian that are ideal for web-based applications.  **Licensing Costs:** Linux is open-source and free to use, which means there are no licensing costs associated with the operating system for the client. | **Characteristics:** Windows is a widely used operating system with a large user base and extensive software support. It offers a range of editions suitable for different scenarios.  **Advantages:** Windows provides a familiar environment for users and developers. It has strong support for various web development technologies and frameworks.  **Weaknesses:** Windows hosting may require more resources compared to Linux for the same workload. Licensing costs can be a concern for large-scale deployments.  **Server-Based Deployment:** Windows Server editions are specifically designed for server deployments and can host web-basedapplications effectively.  **Licensing Costs:** Windows Server requires licenses, and the client would need to consider the costs based on the number of servers and cores required for their scale of deployment. | **Characteristics:** Mobile platforms (iOS and Android) are designed for mobile devices like smartphones and tablets. They are not traditional server operating systems.  **Advantages**: Mobile platforms can be utilized for client-side applications or mobile game versions of Draw It or Lose It. They have extensive user reach in the mobile market.  **Weaknesses:** Mobile devices are not suitable for server-based deployments as they lack the necessary hardware and infrastructure.  **Server-Based Deployment:** Mobile platforms are not meant for server hosting, so they would not be used for the main server-side deployment.  **Licensing Costs:** Mobile platforms do not require licensing costs for server hosting since they are not used for server-side deployments. |
| **Client Side** | **Cost:**  - Higher cost due to hardware and software expenses specific to the Mac platform  **Time:**  - Vary based on application complexity and platform-specific optimizations  **Expertise:**  - macOS SDK, Xcode, Interface Builder, understanding of macOS design principles, and experience with app submission to the Mac App Store. | **Cost:**  - Linux might have higher development costs compared to Mac and Windows due to various distributions and hardware configurations, requiring more testing and compatibility adjustments.  **Time:**  - Vary based on application complexity and platform-specific optimizations  **Expertise:**  - Familiarity with various Linux distributions, Linux-specific APIs, libraries, and packaging tools.. | **Cost:**  - Cost may vary based on licensing fees for Windows development tools and technologies  **Time:**  - Vary based on application complexity and platform-specific optimizations  **Expertise:**  - Windows SDK, Visual Studio, XAML (for UWP), WinForms/WPF (for traditional applications), and knowledge of Windows-specific APIs. | **Cost**:  - The cost can be relatively higher due to separate codebases for each platform, requiring more development and testing effort.  **Time:**  - Vary based on application complexity and platform-specific optimizations  **Expertise:**  Adequate expertise in mobile app development, platform-specific languages (Swift, Kotlin), and frameworks is crucial |
| **Development Tools** | **Programming Languages:**  - Java, Swift and Objective-C.  **IDEs:**  - Xcode, IntelliJ IDEA, Eclipse, Xojo.  **Additional Tools:**  - Cocoa and Cocoa Touch frameworks.  **Impact on Development Team:**  - Specialization of developers for each platform may be needed.  - Requires effective communication and collaboration for integration.  - Increased complexity due to platform-specific code and guidelines.  - Longer development cycles for multiple platforms.  - Some tools may have associated licensing fees (e.g., IntelliJ IDEA). | **Programming Languages:**  - C, C++, Python, Ruby, and Java.  **IDEs:**  - Eclipse, JetBrains IDEs, and Visual Studio Code.  **Additional Tools:**  - GCC/G++ compilers, GNU Make, and Git.  **Impact on Development Team:**  - Understanding Linux-specific considerations (file paths, libraries).  - Familiarity with Git for code management and collaboration.  - Comfort with open-source culture and contributing to projects.  - Strong command-line skills for Linux development tasks. | **Programming Languages:**  - C#, VB.NET, Java, .NET Framework and C++.  **IDE**:  - Visual Studio, Visual Studio Code, Eclipse, IntelliJ IDEA.  **Additional Tools:** - WPF and UWP frameworks.  **Impact on Development Team:**  - Understanding Windows-specific considerations (WinAPI, .NET framework).  - Knowledge of package managers like Chocolatey for software management.  - Familiarity with Git or Microsoft's Team Foundation Version Control (TFVC).  - Coordination with developers specialized in Windows development. | **Programming Languages:**  - iOS: Swift and Objective-C with Xcode as the IDE.  - Android: Kotlin and Java with Android Studio as the IDE.  **Additional Tools:**  - React Native (JavaScript) and Flutter (Dart)  **Impact on Development Team:**  - Diverse skill set required (Java, Kotlin for Android, Swift for iOS).  - Familiarity with multiple IDEs  - Knowledge of cross-platform frameworks  - Understanding mobile platform-specific APIs and design guidelines.  - Familiarity with Git or other version control systems for code collaboration.  - Knowledge of app store submission processes for Android and iOS.  - Optimization techniques for mobile device performance and battery efficiency. |

## 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 support the expansion of Draw It or Lose It to different computing environments, I recommend using a web-based platform. This means the game application can be accessed and run on various operating systems like Linux, Mac and especially Windows. By adopting a Windows architecture, the game becomes more flexible, compatible, and accessible to users across different platforms. It eliminates the need for developing and deploying platform-specific versions. Additionally, a Windows platform allows for easier scalability, maintenance, and updates, making it a practical choice for reaching a wider audience.

1. **Operating Systems Architectures**:

The chosen operating platform architecture for Windows utilizes a web-based platform, leveraging standard web technologies such as HTML, CSS, and JavaScript for the user interface. The game logic and data processing occur on the server side. Users interact with the game through web browsers, which communicate with the server using HTTP requests and responses. On the server side, a backend framework compatible with Windows, such as ASP.NET or .NET Core, is utilized to handle the game logic, manage game sessions, and process user inputs. The backend interacts with the Windows-specific database system, such as Microsoft SQL Server, to store and retrieve game-related information. This architecture ensures compatibility with the Windows operating system, providing a user-friendly and seamless experience for Windows users without requiring platform-specific development or installations. Additionally, it offers scalability, simplified maintenance, and the ability to reach a broader audience across different platforms.

1. **Storage Management**:

An appropriate storage management system to be used with the recommended web-based platform on Windows would be Microsoft SQL Server. Microsoft SQL Server is a relational database management system (RDBMS) specifically designed for Windows operating systems. It offers robust features for managing and storing structured data efficiently. With Microsoft SQL Server, developers can create and manage databases to store game-related information securely. It provides reliable transactional support, scalability, and high performance, making it suitable for handling the data processing requirements of the Draw It or Lose It game. Additionally, Microsoft SQL Server integrates seamlessly with Windows technologies, including the .NET framework and Windows Server, ensuring optimal compatibility and efficient data management for the application.

1. **Memory Management**:

Windows can incorporate memory management techniques to ensure efficient memory usage for the Draw It or Lose It software. Windows utilizes virtual memory, allowing the operating system to allocate memory beyond physical RAM by utilizing disk space. This provides an expanded memory space and enhances performance. Additionally, Windows employs garbage collection, an automatic memory management technique, to identify and reclaim memory that is no longer in use by the application. This prevents memory leaks and optimizes memory usage. By leveraging these memory management techniques, the Draw It or Lose It software on Windows can effectively allocate, utilize, and release memory resources, resulting in improved performance, stability, and responsiveness throughout the game.

1. **Distributed Systems and Networks**:

To make Draw It or Lose It work on different platforms, like Windows, it needs a special way of communicating between them. This is done through a distributed software system and a reliable network. The distributed software system spreads different parts of the game across the platforms, allowing them to talk to each other smoothly. They use standard protocols, like HTTP or WebSocket, to exchange information in real-time. To handle any problems with the network, backup plans are put in place to keep the game running even if there are issues. Making sure the game works well on different platforms means it needs to be compatible with all of them. By using a distributed software system and a strong network, Draw It or Lose It can communicate between platforms, like Windows, and provide a fun and consistent gaming experience for players.

1. **Security:**

To protect user information on various platforms, including Windows, Linux, Mac, and mobile devices, several security measures can be implemented. These include strong encryption to safeguard data, robust authentication mechanisms to verify user identities, secure development practices to prevent vulnerabilities, secure network communication through firewalls and VPNs, and implementing logging and monitoring for detecting suspicious activities. Windows provides built-in security features like antivirus software, firewall protection, encryption, and advanced authentication methods. However, it is important to regularly update and configure these features, follow best practices such as strong passwords and user education, and conduct security assessments to enhance the security of Windows. By implementing comprehensive security measures, user information can be effectively protected across platforms, ensuring a safe environment for Draw It or Lose It players.