

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[CS 230 Project Software Design Template](#bookmark) 1

[Table of Contents](#bookmark1)**2**

[Document Revision History](#bookmark2)**2**

[Executive Summary](#bookmark3)**3**

[Requirements](#bookmark4)**3**

[Design Constraints](#bookmark5)**3**

[System Architecture View](#bookmark6)**3**

[Domain Model](#bookmark7)**3**

[Evaluation](#bookmark8)**4**

[Recommendations](#bookmark9)**5**

## [Document Revision History](#bookmark10)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | 02/23/25 | Jeremiah Longwa | Final prototype design for Draw It or Lose It Software. |

**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](#bookmark11)

The software design problem involves creating a web-based game application called "Draw It or Lose It," inspired by the 1980s television game "Win, Lose or Draw." The game will feature teams competing to guess what is being drawn from a library of stock images within a set time limit. The solution proposed includes developing a scalable, secure, and user-friendly application using modern web technologies and cloud infrastructure. The client must understand the importance of unique identifiers for games, teams, and players to ensure proper game management and prevent conflicts.

## Requirements

Business Requirements:

1. Engaging User Experience: The game must be fun and engaging to attract and retain users.

2. Monetization: The application should have features that allow for monetization, such as in-app purchases or advertisements.

3. Branding: The game should reflect the brand identity and values of the client.

4. Market Reach: The game should be accessible to a wide audience across different devices and platforms.

5. User Retention: Features that encourage repeat usage, such as leaderboards, rewards, and social sharing, should be included.

6. Customer Support: There should be mechanisms for users to get help and support when needed.

Technical Requirements:

1. Scalability: The application must support a large number of concurrent users without performance issues.

2. Low Latency: Ensure real-time interaction with minimal delay.

3. Cross-Platform Compatibility: The game must work seamlessly on desktops, tablets, and smartphones.

4. Security: Implement strong security measures to protect user data and game integrity.

## [Design Constraints](#bookmark12)

1. Scalability: The game must support multiple concurrent users and teams without performance degradation. This requires a scalable backend architecture, likely leveraging cloud services that can dynamically allocate resources based on user demand.

2. Latency: Low latency is crucial for a real-time gaming experience. The application design should incorporate efficient data transmission protocols and potentially use edge computing to minimize the distance between users and servers.

3. Cross-Platform Compatibility: The game should be accessible from various devices, including desktops, tablets, and smartphones. This necessitates a responsive design and thorough testing across different browsers and operating systems.

4. Security: Protecting user data and ensuring secure gameplay is paramount. The design must include encryption, secure authentication mechanisms, and regular security audits to safeguard user information and maintain game integrity.

5. Unique Identifiers: Each game, team, and player must have unique identifiers to prevent conflicts and ensure that only one instance of the game exists in memory at any given time. This will involve implementing a robust system for generating and managing these identifiers.

Implications of Design Constraints:

- Scalability: Requires investment in cloud infrastructure and possibly a micro services architecture to ensure the application can grow with the user base.

- Latency: May necessitate the use of content delivery networks (CDNs) and optimization of server locations to ensure low-latency connections.

- Cross-Platform Compatibility: Increases the complexity of development and testing, as the application must be thoroughly vetted on multiple devices and browsers.

- Security: Demands ongoing attention to potential vulnerabilities, regular updates, and compliance with data protection regulations.

- Unique Identifiers: Requires a systematic approach to generate and manage unique identifiers for games, teams, and players, ensuring proper game state management and preventing conflicts.

## [System Architecture View](#bookmark13)

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](#bookmark14)

The UML diagram shows the relationships between the classes in a gaming application. The diagram shows that the GameService class is responsible for managing games, and the ProgramDriver class uses the GameService to manage games. The diagram also shows that the Game, Team, and Player classes are all subclasses of the Entity class.

Classes and Relationships

1. Entity (abstract class)

- Attributes: id, name

- This class serves as a base class for other entities in the system.

2. Game (inherits from Entity)

- Attributes: name

- Relationships:

- One-to-Many relationship with Team(A game can have multiple teams)

3. Team (inherits from Entity)

- Attributes: name

- Relationships:

- One-to-Many relationship with Player(A team can have multiple players)

4. Player (inherits from Entity)

- Attributes: id, name

5. GameService

- Attributes: games (list of games), nextGameId, nextPlayerId, nextTeamId, serviceInstance

- This class is responsible for managing games, teams, and players.

6. ProgramDriver

- Uses GameService to manage games.

Object-Oriented Programming Principles Demonstrated

1. Inheritance:

- The Entity class is an abstract base class from which Game, Team, and Player inherit. This demonstrates the principle of inheritance, allowing for code reuse and a hierarchical relationship between classes.

2. Encapsulation:

- Each class encapsulates its own attributes and behaviors. For example, GameService manages the list of games and IDs internally, providing a clear interface for other classes to interact with it.

3. Abstraction:

- The Entity class is abstract, meaning it cannot be instantiated on its own. It provides a common interface and shared attributes (`id` and `name`) for its subclasses, abstracting away the specific details of each entity type.

4. Polymorphism:

- Through inheritance, polymorphism is achieved. Methods that operate on Entity objects can also operate on Game, Team, and Player objects, allowing for flexible and interchangeable use of these objects.

Efficiency in Software Requirements

- Modularity:

- The separation of Game, Team, and Player classes allows for modular design. Each class handles its own specific responsibilities, making the system easier to maintain and extend.

- Scalability:

- The use of lists to manage multiple games, teams, and players allows the system to scale efficiently as more entities are added.

- Singleton Pattern:

- The serviceInstance attribute in GameService suggests the use of the Singleton pattern, ensuring that only one instance of GameService exists. This is efficient for managing shared resources and providing a global point of access.

**"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](#bookmark15)

Evaluation of the characteristics, advantages, and weaknesses of each operating platform for hosting a web-based software application and for supporting multiple client types.

Server Side Evaluation

Linux

Characteristics:

- Open-source and widely used for server hosting.

- Highly customizable and flexible.

Advantages:

- Cost-effective: No licensing fees.

- Stability and reliability: Known for its uptime and performance.

- Security: Strong security features and frequent updates.

- Scalability: Easily scalable to handle thousands of users.

Weaknesses:

- Requires expertise: May need specialized knowledge to set up and manage.

- Compatibility: Some proprietary software may not be compatible.

Server-Based Deployment:

- Yes, Linux offers robust server-based deployment options.

Licensing Costs:

- Free, open-source software.

Mac (macOS)

Characteristics:

- Unix-based operating system with a strong graphical interface.

Advantages:

- User-friendly interface.

- Strong security features.

- Integration with other Apple products.

Weaknesses:

- High cost: Expensive hardware and potential licensing fees.

- Limited server options: Not as widely used or supported for server hosting as Linux or Windows.

Server-Based Deployment:

- Yes, but not commonly used for large-scale web hosting.

Licensing Costs:

- High, due to the cost of Apple hardware and potential software licenses.

Windows

Characteristics:

- Widely used with extensive support for various software applications.

Advantages:

- User-friendly interface.

- Wide range of compatible software.

- Strong support and documentation.

Weaknesses:

- Licensing costs: Requires purchasing licenses.

- Security: More susceptible to malware and viruses compared to Linux.

Server-Based Deployment:

- Yes, widely used for server hosting, especially with IIS (Internet Information Services).

Licensing Costs:

- High, due to the cost of Windows Server licenses.

Client Side Evaluation

Software Development Considerations

Compatibility Requirements:

- Ensure the application is built using responsive web design principles.

- Use standard web technologies like HTML5, CSS3, and JavaScript.

- Test across multiple browsers (Chrome, Firefox, Safari, Edge) and devices (iOS, Android).

Development Tools:

- Integrated Development Environments (IDEs): Visual Studio Code, IntelliJ IDEA, etc.

- Frameworks and Libraries: React, Angular, Vue.js for front-end; Node.js, Django, Flask for back-end.

- Testing Tools: Selenium, BrowserStack for cross-browser testing.

Cost, Time, Expertise:

- Cost: Varies depending on the tools and technologies used. Open-source tools can reduce costs.

- Time: Building a responsive, cross-platform application can be time-consuming. Proper planning and project management are essential.

- Expertise: Requires skilled developers familiar with web technologies, responsive design, and cross-platform compatibility.

Development Tools

IDEs and Editors:

- Visual Studio Code, IntelliJ IDEA, Sublime Text.

Frameworks:

- Front-end: React, Angular, Vue.js.

- Back-end: Node.js, Django, Flask.

Testing Tools:

- Selenium, BrowserStack, Jest for unit testing.

Conclusion

- Server Side: Linux is the most cost-effective and scalable option for hosting the web application. Windows is also a viable option but comes with higher licensing costs. macOS is less commonly used for server hosting and could be more expensive.

- Client Side: Developing a responsive web application using standard web technologies ensures compatibility across all platforms. Proper planning, use of modern development tools, and thorough testing are essential to support multiple types of clients effectively.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a user-friendly operating system with a strong focus on design and aesthetics. It is known for its stability and security, making it a good choice for hosting web-based applications. However, Mac can be more expensive than other operating systems, and its market share is smaller than Windows or Linux. | Linux is a powerful and flexible operating system that is known for its open-source nature and its ability to be customized. It is a popular choice for hosting web-based applications due to its low cost and high performance. However, Linux can be more complex to manage than other operating systems, and it may require more technical expertise. | Windows is the most popular operating system in the world, and it is known for its ease of use and its wide range of software compatibility. It is a good choice for hosting web-based applications, but it can be more expensive than other operating systems, and it may be more vulnerable to security threats. | Mobile devices are becoming increasingly popular for accessing web-based applications. They offer a number of advantages, such as portability and convenience. However, mobile devices can be more limited in terms of processing power and memory, and they may have different security considerations than other operating systems. |
| **Client Side** | When developing software for Mac, it is important to consider the specific needs of Mac users. For example, Mac users are often more tech-savvy than Windows users, and they may expect a more polished and user-friendly experience. | When developing software for Linux, it is important to consider the wide range of Linux distributions that are available. Each distribution has its own unique set of features and capabilities, so it is important to choose the right distribution for your target audience. | When developing software for Windows, it is important to consider the large number of Windows users who may have different levels of technical expertise. It is also important to consider the different versions of Windows that are available, as each version may have different features and capabilities. | When developing software for mobile devices, it is important to consider the different screen sizes and resolutions that are available. It is also important to consider the different operating systems that are used on mobile devices, such as Android and iOS. |
| **Development Tools** | When developing software for Mac, it is important to use the right development tools. There are a number of popular development tools that are available for Mac, such as Xcode and Visual Studio Code. | When developing software for Linux, it is important to use the right development tools. There are a number of popular development tools that are available for Linux, such as GCC and GDB. | When developing software for Windows, it is important to use the right development tools. There are a number of popular development tools that are available for Windows, such as Visual Studio and Visual Studio Code. | When developing software for mobile devices, it is important to use the right development tools. There are a number of popular development tools that are available for mobile devices, such as Android Studio and Xcode. |

## 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" to other computing environments, I recommend using a cross-platform framework such as Unity. Unity supports multiple operating systems, including Windows, macOS, Linux, iOS, and Android. This will allow The Gaming Room to develop the game once and deploy it across various platforms with minimal changes.

2. **Operating Systems Architectures:**

Unity utilizes a component-based architecture, which is highly modular and flexible. This architecture allows developers to create reusable components that can be easily integrated into different parts of the game. Unity's architecture supports both 32-bit and 64-bit systems, ensuring compatibility with a wide range of devices. Additionally, Unity's scripting is primarily done in C#, which is a powerful and widely-used programming language.

3. **Storage Management:**

For storage management, I recommend using a cloud-based storage solution like Firebase. Firebase provides real-time database capabilities, which are essential for a multiplayer game like "Draw It or Lose It." It also offers easy integration with Unity and supports data synchronization across multiple devices. This ensures that game data is consistently updated and accessible from any platform.

4. **Memory Management:**

Unity uses an automatic memory management system with garbage collection to handle memory allocation and deallocation. This helps prevent memory leaks and ensures efficient use of memory resources. Unity also provides tools like the Profiler to monitor and optimize memory usage, which is crucial for maintaining performance, especially on mobile devices.

5. **Distributed Systems and Networks:**

To enable communication between various platforms, "Draw It or Lose It" can utilize Unity's built-in networking capabilities or third-party solutions like Photon. These systems support real-time multiplayer functionality and can handle the complexities of distributed systems, such as connectivity issues and data synchronization. Using a cloud-based server infrastructure can help manage the dependencies between components and ensure seamless communication even during outages.

6. **Security:**

Security is paramount, especially when handling user information. Unity supports encryption for data transmission, ensuring that user data is protected during communication between platforms. Additionally, using Firebase for storage adds another layer of security, as it provides robust authentication and authorization mechanisms. Implementing secure coding practices and regularly updating the software to patch vulnerabilities will further enhance security.