

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 | 07/11/2023 | Darren Sheftic | Initial Design Proposal |

## [Executive Summary](#_sbfa50wo7nsh)

This document outlines the software design for the web-based Draw It or Lose It game application for the company “The Gaming Room”. The game is based on the concept of the popular 1980s television game "Win, Lose or Draw," where teams compete to guess a puzzle based on rendered images. The application will allow multiple teams with multiple players to participate in four rounds of play. To ensure uniqueness and prevent conflicts the game and team names must be unique as well as the application will support only one instance of the game in memory at any given time.

The proposed solution involves the development of a web-based distributed environment for the game application. This will allow users to access and play the game through a web browser without the need for installation. The game application will utilize a large library of stock drawings to generate clues for the players to guess. The application will implement a timer mechanism to manage the duration of each round and provide equal opportunities for teams to guess the puzzle. By following this design, the Draw It or Lose It game can be effectively developed and deployed in a user-friendly manner.

## Requirements

1. Business Requirements:
   * *The application should be a web-based Draw It or Lose It game.*
   * *The game should be based on the concept of the popular 1980s television game "Win, Lose or Draw."*
   * *Multiple teams with multiple players should be able to participate in four rounds of play.*
   * *Unique game and team names must be enforced to prevent conflicts and ensure uniqueness.*
   * *The application should support only one instance of the game in memory at any given time.*
2. Technical Requirements:
   * *The game application should be web-based, accessible through web browsers.*
   * *It should conform to web standards to ensure cross-platform compatibility.*
   * *A registration and authentication process is needed to enforce unique game and team names.*
   * *The application must have a mechanism to manage game instances and prevent conflicts.*
   * *A large library of stock drawings should be included as clues for the players.*
   * *Each round of the game should be limited to one minute, with a timer mechanism to manage round duration.*
   * *The application should have a user-friendly interface and smooth gameplay experience.*
   * *The UML class diagram should be implemented to represent classes and their relationships accurately.*

## [Design Constraints](#_2et92p0)

1. Web-Based: The game application will be developed as a web-based application. This constraint limits the technologies and frameworks that can be used in the development process. The application must be compatible with web browsers and conform to web standards for ensuring cross-platform compatibility.
2. Unique Game and Team Names: The game requires unique game and team names that imposes limitations on the registration and authentication process. The application must include a way to check the instance of names.
3. Single Instance in Memory: The game application should allow only one instance of the game to be active in memory at any time. This requires the implementation of a way to manage the game instances and prevent multiple instances from conflicting with each other. It also implies the need for unique identifiers to differentiate between instances of games, teams, and players.
4. Stock Drawings Library: The game application will rely on a large library of stock drawings to generate clues for the players. This constraint requires the inclusion of the drawing library within the application's resources.
5. Timer Mechanism: Each round of the game is limited to one minute, with drawings rendered at a steady rate and fully completed at the 30-second mark. This constraint necessitates the implementation of a timer to track and manage the duration of each round. The timer should be synchronized with the rendering of drawings.
6. User Interface and User Experience: The game application should have a user-friendly interface to create smooth gameplay. The interface should allow players to interact with the game easily, guess the puzzles, and view relevant information.

## [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 class diagram provided represents a software design that includes several classes and their relationships.

ProgramDriver Class:

The ProgramDriver class is at the top of the diagram and represents the starting point of the program. It contains a single method, main(), which serves as the launch point for the application.

SingletonTester Class:

The SingletonTester class is a class that represents a tester class responsible for testing the Singleton pattern. It contains a method, testSingleton(), which tests the implementation.

Entity Class:

The Entity class represents an entity in the system and includes properties such as id (long) and name (String). It has a constructor to initialize these properties and getter methods to access them. The toString() method is also provided to return a string of the Entity object.

GameService Class:

The GameService class represents a class responsible for managing games in the system. It maintains a list of Game objects and keeps track of identifiers for the next game, player, and team to be created. It follows the Singleton pattern, as indicated by the <<uses>> relationship from ProgramDriver to SingletonTester. This design ensures that only one instance of GameService exists throughout the application.

Game Class:

The Game class represents a game in the system. It maintains a list of Team objects and includes a constructor to initialize the game with an id and a name. The addTeam() method allows adding teams to the game, and the toString() method returns a string representation of the Game object.

Team Class:

The Team class represents a team in the system. It maintains a list of Player objects and includes a constructor to initialize the team with an id and a name. The addPlayer() method allows adding players to the team, and the toString() method returns a string representation of the Team object.

Player Class:

The Player class represents a player in the system. It includes a constructor to initialize the player with an id and a name. The toString() method returns a string representation of the Player object.

Encapsulation Object Oriented Principle:

Encapsulation is demonstrated using private and public access modifiers on class members. By encapsulating variables and methods within classes, the diagram encourages controlled access to the class members.

Singleton Object Oriented Principle:

The GameService class demonstrates the Singleton pattern by ensuring that only one instance of the class exists. This is achieved by having a private constructor and providing a getInstance() method that returns the single instance of the GameService class. The Singleton pattern helps ensure that there is a unique instance responsible for managing the game.

**"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** | Characteristics:  macOS is a Unix-based system, providing a stable and reliable platform for hosting web applications.  Advantages:  Integration with other Apple services and products (e.g., iCloud, Xcode).  Xcode IDE offers development tools and simulator for iOS testing.  Weaknesses:  Limited market share compared to Windows and Linux.  Costly hardware options.  Licensing restrictions and limitations.  Additional Components or Options Needed:  macOS Server, an add-on software package that provides additional server functionality and tools. | Characteristics:  Linux is open-source and highly customizable, suitable for hosting web applications.  Advantages:  Strong security features and permissions control.  Broad hardware support makes it versatile.  Weaknesses:  There is no single, standardized version of the operating system that everyone uses.  Some software may be exclusive and unavailable.  Additional Components or Options Needed:  Linux distros typically do not come with pre-installed web server software. Developers may need to choose and install web server software like Apache, Nginx, or others based on their requirements. | Characteristics:  Windows Servers provides a variety of software options for hosting web-based applications.  Advantages:  Windows is widely used, providing a familiar experience for many users.  Strong support for various programming languages and development frameworks.  Weaknesses:  The wide range of devices and operating system deviations across Windows can lead to challenges in compatibility and testing.  Limited and poor resource management of the operating system can affect performance due to background processes, lack of optimization, and crashing under heavy usage. | Characteristics:  Mobile devices have limited resources, including processing power, memory, and storage.  Advantages:  Large user base allows for broad reach to potential users.  App marketplaces allow for visibility, transparency, and easy distribution.  Built-in features for GPS, camera, various other functions.  Weaknesses:  Separate frameworks, languages, and development tools needed for different mobile operating systems (primarily iOS and Android)  Constant updates needed to configure to new versions of mobile operating systems.  App Store approval process  Challenging to create consistent experience with different screen sizes, resources, and operating system versions.  Cannot host a web-based application. |
| **Client Side** | Cost:  Developing and testing for Mac-based clients may require additional resources and time due to the need for macOS specific expertise.  Compatibility: Ensuring that the application functions as intended on various macOS versions and hardware is important.  Expertise:  Developers with macOS expertise may be easier to find compared to mobile and Linux platforms.  Skills Needed to Meet Software Requirements:  Developers should be skilled in macOS app development using programming languages like Swift and Objective-C.  Security:  macOS is known for its strong security features, but it also requires careful configuration. | Cost:  Developing and testing for Linux-based clients may require fewer resources compared to other platforms.  Compatibility:  Ensuring cross-platform compatibility for different Linux distributions can be challenging.  Expertise:  Developers with Linux expertise may be easier to find than mobile platforms.  Skills Needed to Meet Software Requirements:  For supporting multiple types of clients on Linux, developers should have proficiency in cross-platform web development using web frameworks like HTML, CSS, and JavaScript. | Cost:  Developing and testing for Windows-based clients may involve additional licensing costs and hardware variations.  Compatibility:  Most compatible operating system due to prevalence.  Expertise:  Windows-oriented expertise is more prevalent than other platforms.  Skills Needed to Meet Software Requirements:  Universal Windows Platform (UWP) concepts for building Windows applications that work across various devices. | Cost:  Developing and testing for various mobile platforms can be resource-intensive and time-consuming.  Compatibility:  Web-based applications can be accessed from various mobile devices regardless of the operating system. By building cross-platform web applications developers can ensure compatibility across a wide range of mobile devices.  Expertise:  Mobile devices have unique interface requirements and limitations that must be considered during design and development.  Skills Needed to Meet Software Requirements:  Familiarity with Cloud-based Services  Mobile App Development based languages |
| **Development Tools** | Swift and Objective-C for iOS/macOS app development.  JavaScript, HTML, CSS for web development.  Xcode IDE for macOS app development.  Tomcat for web server hosting. | Visual Code Studio, IntelliJ, Eclipse, Apache Netbeans, etc for IDEs  Python, Node.js, Ruby, PHP, Java, Go, etc., are languages for web app development. | C#, .NET, Visual Studio for app development.  JavaScript, HTML, CSS for web development.  Visual Studio Code, JetBrains Rider, Eclipse, etc., IDEs for web development. | Swift and Xcode IDE for iOS  Kotlin or Java and Android Studio IDE for Android.  For cross-platform mobile app development:  React Native, .NET, Flutter frameworks can be used. |

## 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**: Based on the requirements for expanding Draw It or Lose It to other computing environments and ensuring cross-platform compatibility, I recommend adopting a cloud-based operating platform and using Linux as the operating system for the server-side of the Draw It or Lose It web-based application. Cloud platforms offer numerous advantages, including scalability, flexibility, and accessibility from various devices and locations. By utilizing cloud services, The Gaming Room can easily deploy the game application to different operating systems without significant modifications. Linux, being an open-source platform, allows The Gaming Room to avoid licensing fees and reduces operational expenses. Additionally, Linux has a reputation for strong security, making it a reliable choice to protect user data and the application from potential cyber threats. Finally, Linux offers excellent support for web-based applications, including web servers, databases, and programming languages commonly used for web development. It seamlessly integrates with various web technologies, enabling efficient handling of HTTP requests and real-time communication, which is essential for interactive gameplay in Draw It or Lose It.
2. **Operating Systems Architectures**: Cloud platforms typically support various operating system architectures, including Windows, Linux, and macOS. The choice of the operating system architecture will depend on the specific cloud service provider selected by The Gaming Room. Popular cloud service providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer support for multiple operating system environments, allowing the game application to run seamlessly on different platforms. Linux follows a monolithic kernel architecture, which means the kernel directly interacts with the hardware and handles essential tasks like memory management, process scheduling, and device drivers. It provides strong performance and stability due to the tight integration of kernel components. A kernel is the main component of an operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible.
3. **Storage Management**: For the recommended cloud-based operating platform, The Gaming Room can utilize cloud-based storage services. Cloud storage services provide scalable and cost-effective solutions for managing large libraries of stock drawings used as clues in the game. AWS Simple Storage Service (S3), Azure Blob Storage, and Google Cloud Storage are examples of cloud storage solutions that can efficiently handle the storage requirements of the Draw It or Lose It application.
4. **Memory Management**: Cloud platforms handle memory management automatically and efficiently. The infrastructure of a cloud provider manages memory allocation and deallocation, ensuring optimal performance and resource utilization for the Draw It or Lose It software. The Gaming Room will not have to worry about manual memory management, as the cloud platform will take care of these aspects. On top of that Linux manages memory using virtual memory and demand paging techniques. It allocates virtual memory space to processes, allowing them to operate as if they constantly have memory. When the application requires more memory than physically available, Linux then uses demand paging to move less used memory pages to disk temporarily. This ensures efficient memory utilization and allows the Draw It or Lose It application to handle various user sessions without memory-related performance issues.
5. **Distributed Systems and Networks**: To enable communication between various platforms, the Draw It or Lose It game application can utilize RESTful APIs in the cloud platform. These APIs allow different components of the game to communicate seamlessly over the network. The client will communicate with the server using HTTP requests. The Draw It or Lose It game application will utilize RESTful APIs to allow clients to interact with the server. These browsers on users' computers or mobile devices will send HTTP requests to the server's endpoints and the server will respond with appropriate data or actions. The distributed nature of cloud services ensures that instances of the game can be deployed across different regions, reducing the impact of connectivity outages or localized issues. The cloud provider's infrastructure automatically manages load balancing and failover, which is a method of protecting computer systems from failure where standby equipment automatically takes over when the main system fails, enhancing the overall reliability of the system.
6. **Security**: Security is a top priority for the client and developers. Cloud providers implement various security measures, including data encryption, access controls, role and access management, and regular security audits. The Gaming Room can utilize these built-in security features to safeguard user information across platforms. Additionally, the cloud platform integrates with third-party security solutions enhancing the overall security of applications.

By implementing the following security measures The Gaming Room can ensure a secure and reliable game application across multiple platforms and browsers:

* Encrypted communication: Secure data in transit by using HTTPS to encrypt communication between clients and the server.
* Authentication: Implement user authentication procedures to allow only authorized users access to the game.
* Authorization: Enforce role-based access control to restrict a user’s actions based on their role (players, administrators).
* Input validation: Validate all user inputs to prevent common security vulnerabilities like SQL injection.
* Data encryption: Safeguard sensitive data by encrypting it, both when stored in the database and during transmission.
* Rate limiting: Implement rate limiting to prevent brute-force attacks and protect against excessive API requests.
* Security audits: Conduct regular security audits and testing to identify and fix potential vulnerabilities.