

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 | 03/23/2024 | Nick Giesen | Initial Report |
| 1.1 | 04/05/2024 | Nick Giesen | Adjusted Evaluation table layout |
| 1.2 | 04/20/2024 | Nick Giesen | Adjusted readability layout for Recommendations |

**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 challenge at hand is to develop a web-based game application called "Draw It or Lose It" which is akin to the 1980s television game "Win, Lose or Draw." In this game, teams compete to guess what is being drawn, but instead of players manually drawing on an easel, the application renders images from a stock library as clues. The game comprises four rounds, each lasting one minute, with drawings being fully rendered by the 30-second mark. If a team fails to guess the puzzle in time, other teams get a chance to guess within a 15-second time limit.

## Requirements

To meet the client's requirements, the software design will include the ability to accommodate multiple teams, with each team comprising several players. Game and team names will be unique to avoid confusion, and unique identifiers will be assigned to each instance of a game, team, or player. This ensures that only one instance of the game exists in memory at any given time.

## [Design Constraints](#_2et92p0)

Web-Based Distributed Environment: Developing the game application in a web-based distributed environment imposes constraints related to scalability, latency, and security. To address these constraints, the application architecture must support efficient communication between clients and servers, handle concurrent user interactions, and implement security measures to protect user data and prevent unauthorized access.

Unique Game and Team Names: Ensuring that game and team names are unique presents a challenge in managing data integrity and avoiding naming conflicts. This constraint requires implementing validation mechanisms during the registration process to check for name availability and prevent duplication.

Single Instance in Memory: Limiting the game to a single instance in memory necessitates designing the application with a state management mechanism to control access and ensure data consistency. This constraint may influence decisions regarding session management, caching strategies, and resource allocation to optimize performance and resource utilization.

Real-Time Rendering and Timer Management: The requirement for real-time rendering of stock drawings and precise timer management adds complexity to the application logic. To meet these demands, the application must employ efficient rendering techniques, handle timer events accurately, and synchronize actions across multiple clients to maintain gameplay consistency.

## [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)

**ProgramDriver Class**: This class represents the entry point of the program and contains the main method. It serves as the starting point for the application's execution.

**SingletonTester Class**: This class is responsible for testing the singleton implementation of the GameService class.

**Entity Class**: This class serves as a base class for other classes in the system and contains common attributes such as id and name. It also provides methods for accessing and modifying these attributes.

**GameService Class**: This class manages the lifecycle of games and provides services for creating, accessing, and manipulating game objects. It follows the Singleton design pattern to ensure that only one instance of the class exists throughout the application.

**Game Class**: Represents a game within the system. It contains a list of teams participating in the game and provides methods for adding teams and generating a string representation of the game.

**Team Class**: Represents a team participating in a game. It contains a list of players belonging to the team and provides methods for adding players and generating a string representation of the team.

**Player Class**: Represents a player participating in a team. It provides methods for accessing player attributes and generating a string representation of the player.

The relationships between the classes demonstrate several object-oriented programming principles:

**Encapsulation**: Each class encapsulates its data and behavior, providing clear interfaces for interacting with the objects. For example, the GameService class encapsulates game-related operations, such as adding games and retrieving game counts.

**Inheritance**: The Entity class serves as a base class for other classes in the system, demonstrating inheritance. This allows subclasses (e.g., Game, Team, Player) to inherit common attributes and methods from the Entity class.

**Association**: The diagram illustrates associations between classes, such as GameService being associated with Game, Game being associated with Team, and Team being associated with Player. These associations represent how objects of one class are related to objects of another class within the system.

**Singleton Pattern**: The GameService class demonstrates the Singleton design pattern, ensuring that only one instance of the class exists in memory at any given time. This pattern is useful for managing shared resources and centralizing access to services within an 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)

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: Mac servers offer stability, security, and compatibility with various web technologies. They are known for their user-friendly interface and robust performance.  Advantages: Mac servers integrate well with other Apple products and development tools, making them suitable for Apple-centric environments. They also have strong support for web development frameworks like Ruby on Rails and Node.js.  Weaknesses: Mac servers may have limited scalability compared to Linux or Windows servers. Additionally, they are typically more expensive than Linux-based servers. | Characteristics: Linux servers are renowned for their stability, security, and flexibility. They provide excellent support for open-source software and technologies.  Advantages: Linux servers are highly customizable and can be tailored to specific needs. They offer robust performance and scalability, making them ideal for hosting web-based applications.  Weaknesses: Linux servers may have a steeper learning curve for users unfamiliar with the Linux operating system. Additionally, certain proprietary software may have limited support on Linux platforms. | Characteristics: Windows servers are known for their ease of use, compatibility with Microsoft technologies, and strong support for .NET development.  Advantages: Windows servers integrate seamlessly with Microsoft products and development tools, making them a preferred choice for organizations invested in the Microsoft ecosystem. They offer extensive support for ASP.NET, C#, and other Microsoft technologies.  Weaknesses: Windows servers may require additional licensing costs for certain features or services. They may also be more vulnerable to security threats compared to Linux servers. | Characteristics: Mobile devices offer portability, accessibility, and a wide range of hardware capabilities. They are designed to run lightweight applications optimized for mobile platforms.  Advantages: Mobile devices provide direct access to a large user base, allowing developers to reach a broad audience. They offer features like touch interfaces, sensors, and location services, enhancing the user experience.  Weaknesses: Mobile devices have limited processing power and memory compared to desktop or server systems. Developing web-based software for mobile devices requires careful optimization for performance and resource usage. |
| **Client Side** | Considerations: Supporting multiple types of clients on Mac may require expertise in macOS development, knowledge of Apple's development frameworks (e.g., Swift, SwiftUI), and familiarity with Xcode as the primary IDE.  Cost: Development for Mac clients may involve the cost of Apple hardware (e.g., MacBooks, iMacs) and developer program membership fees.  Time: Developing and testing software for Mac clients may take additional time due to platform-specific considerations and the need for compatibility testing across different macOS versions.  Expertise: Developers need expertise in macOS development tools, APIs, and user interface design principles to create seamless experiences for Mac users. | Considerations: Supporting multiple types of clients on Linux may require knowledge of cross-platform development frameworks (e.g., Qt, GTK), familiarity with Linux distributions, and proficiency in programming languages commonly used on Linux (e.g., C/C++, Python).  Cost: Development for Linux clients may involve minimal costs as many development tools and frameworks for Linux are open-source and freely available.  Time: Developing software for Linux clients may require additional time for compatibility testing across different Linux distributions and versions.  Expertise: Developers need expertise in Linux development environments, command-line tools, and package management systems to ensure compatibility and optimal performance on Linux systems. | Considerations: Supporting multiple types of clients on Windows may require expertise in Windows development using languages like C#, .NET framework, and knowledge of Windows-specific APIs. Familiarity with Microsoft Visual Studio is essential for Windows development.  Cost: Development for Windows clients may involve the cost of Windows development tools, such as Visual Studio, and potential licensing fees for certain Microsoft technologies.  Time: Developing software for Windows clients may require time for compatibility testing across different Windows versions and form factors (e.g., desktop, tablet, mobile).  Expertise: Developers need expertise in Windows development tools, frameworks, and user interface guidelines to create polished applications for Windows users. | Considerations: Supporting multiple types of clients on mobile devices requires expertise in mobile app development for platforms like iOS (using Swift or Objective-C) and Android (using Java or Kotlin). Developers also need knowledge of mobile development frameworks (e.g., Flutter, React Native) for cross-platform development.  Cost: Mobile app development may involve costs for developer program memberships (e.g., Apple Developer Program, Google Play Developer Console) and potentially device testing.  Time: Developing mobile applications may require additional time for platform-specific optimizations, testing across different devices and screen sizes, and adherence to platform-specific guidelines.  Expertise: Developers need expertise in mobile app development frameworks, platform-specific APIs, and user experience design principles to create engaging and functional mobile applications. |
| **Development Tools** | Programming Languages: Software for deploying on Mac can be developed using languages such as Swift, Objective-C, and C/C++.  IDEs (Integrated Development Environments): The primary IDE for macOS development is Xcode, which provides a comprehensive set of tools for building macOS, iOS, watchOS, and tvOS applications. Xcode includes features like code editing, debugging, interface design, and performance analysis.  Other Tools: Developers may also use tools like CocoaPods and Carthage for managing dependencies, Git for version control, and instruments for performance tuning and optimization. | Programming Languages: Software for deploying on Linux can be developed using languages such as C/C++, Python, Java, and others.  IDEs: Popular IDEs for Linux development include Visual Studio Code, Eclipse, IntelliJ IDEA, and Qt Creator. These IDEs provide features like code editing, debugging, version control integration, and support for various programming languages and frameworks.  Other Tools: Developers may use build systems like CMake, GNU Make, or Apache Maven, along with package managers like APT (Advanced Package Tool) or YUM (Yellowdog Updater, Modified) for managing dependencies and software distribution. | Programming Languages: Software for deploying on Windows can be developed using languages such as C#, .NET, C/C++, and others.  IDEs: The primary IDE for Windows development is Microsoft Visual Studio, which offers a comprehensive suite of tools for building Windows applications. Visual Studio provides features like code editing, debugging, UI design, performance profiling, and support for various programming languages and frameworks.  Other Tools: Developers may use tools like NuGet for package management, Git for version control, WiX Toolset for creating installers, and Microsoft App Center for app distribution and monitoring. | Programming Languages: Software for deploying on mobile devices can be developed using languages such as Swift or Objective-C for iOS, and Java or Kotlin for Android. Cross-platform frameworks like Flutter, React Native, or Xamarin can also be used for building apps that target multiple mobile platforms.  IDEs: For iOS development, Xcode is the primary IDE, providing tools for building iOS, iPadOS, watchOS, and tvOS applications. For Android development, Android Studio is the recommended IDE, offering features like code editing, debugging, layout design, and performance profiling. Cross-platform IDEs like Visual Studio Code with appropriate plugins can be used for cross-platform mobile development.  Other Tools: Developers may use tools like CocoaPods (for iOS), Android SDK and NDK (for Android), Firebase for backend services, and various testing frameworks for automated testing on mobile devices. Additionally, mobile developers often use emulators/simulators for testing apps on different device configurations. |

## 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 while ensuring compatibility and scalability, a cloud-based operating platform like Amazon Web Services (AWS) is recommended. AWS provides a range of services and features suitable for hosting web-based applications, offering flexibility, reliability, and scalability for future expansion.
2. **Operating Systems Architectures**: AWS supports multiple operating system architectures, including Linux, Windows, and various distributions optimized for cloud environments. It utilizes a virtualized infrastructure that abstracts hardware resources and provides scalable computing capacity through instances running on physical servers.
3. **Storage Management**: For storage management within AWS, Amazon Simple Storage Service (S3) can be utilized for storing static assets like stock drawings, while Amazon Relational Database Service (RDS) or Amazon DynamoDB can be used for managing relational or NoSQL databases, respectively. These services offer scalability, durability, and high availability for storing and accessing application data.
4. **Memory Management**: AWS manages memory allocation and utilization within its cloud infrastructure, dynamically allocating resources to instances based on demand. Additionally, AWS offers services like Elastic Load Balancing (ELB) and Auto Scaling to optimize resource usage and ensure consistent performance for applications like Draw It or Lose It.
5. **Distributed Systems and Networks**: Communication between various platforms can be achieved using AWS services like Amazon API Gateway for creating RESTful APIs to facilitate communication between clients and servers. Additionally, AWS Virtual Private Cloud (VPC) allows for the creation of isolated network environments to securely connect different components within the distributed system. Dependencies between components are managed through fault-tolerant design patterns, such as redundant data centers and multi-Availability Zone deployments, to mitigate connectivity issues and outages.
6. **Security**: AWS provides robust security features to protect user information and ensure data privacy. This includes identity and access management (IAM) for controlling access to AWS resources, encryption services like AWS Key Management Service (KMS) for securing data at rest and in transit, and network security features such as AWS Firewall Manager and AWS WAF (Web Application Firewall) for protecting against unauthorized access and cyber threats. Additionally, AWS compliance certifications and adherence to best security practices provide assurance to clients regarding the security of their data on various platforms.