

Draw It or Lose It!

# **CS 230 Project Software Design**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design** 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 5**](#_Toc115077325)

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  1.1  1.2 | 09/20/2024  10/03/2024  10/14/2024 | Beau Zermeno  Beau Zermeno  Beau Zermeno | Summarized client requirements and constraints.  Discussed how various operating platforms fit into the client-server pattern.  Discussed Windows as recommended platform for converting Draw It or Lose It! to a web-based platform. |

## [Executive Summary](#_sbfa50wo7nsh)

The purpose of the software development for the game "Draw It or Lose It" is to create a web-based, distributed application that allows multiple teams to participate in a guessing game. The game will consist of four rounds, each lasting one minute, with the aim of determining the puzzle based on stock image clues rendered on the screen. Key design elements include unique game, team, and player identifiers to ensure that game instances can be efficiently managed and accessible in memory. This design proposal outlines the architectural and object-oriented approach to meeting the client's requirements while ensuring scalability and maintaining reliability.

## Requirements

* The single-instance memory constraint may require server-based architecture capable of managing states without creating duplicate instances.
* The need for real-time synchronization across distributed systems implies that communication protocols should be used.

## [Design Constraints](#_2et92p0)

* The application must be able to support multiple players across different geographic regions, requiring consideration of latency, server load, and data synchronization.
* Only one instance of the game can be stored in memory, requiring careful management of identifiers for game, team, and player objects. This requires the use of a singleton pattern for the game instance.
* Ensure that game and team names are unique adds complexity to the design, as it requires checking against a database or cache of names.
* The steady rendering of images within 30 seconds and the time-sensitive nature of guessing requires accurate data transfer and updates.

## [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 diagram for "Draw It or Lose It" encompasses the primary entities and relationships involved in the game, including the Game, Team, Player, and Entity classes. Each entity encompasses attributes and behaviors that are relevant to their role in the system.

**Object Classes:**

**Entity Class:** Base class that includes common attributes such as id (unique identifier) and name. This allows for consistent management of game objects and ensures that all entities can be accurately identified.

**Game Class:** Manages the game’s state, such as which teams are involved, game timing, and managing rounds. It is inherited from the Entity class and contains methods to start the game, add teams, and track the progress of each round.

**Team Class:** Manages a group of players and keeps track of the team’s progress within the game. It is also inherited from the Entity class, utilizing the unique ID and name attributes to ensure uniqueness.

**Player Class:** Represents each player within a team, containing attributes such as the player’s name and score. Players can interact with the game by submitting guesses.

**Object-Oriented Principles:**

**Encapsulation:** Each class has its own attributes and methods, ensuring that data and behaviors are categorized logically and only exposed as necessary.

**Inheritance:** The Entity class is used as a base class to avoid redundancy, enabling all game objects to share common attributes such as unique IDs and names.

**Polymorphism:** Methods that interact with several types of entities (such as teams and players) may be formulated to work with any subclass of Entity.

**"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** | MacOS is a Unix-based system like Linux, which makes it a secure and secure environment. It includes built-in development tools such as Apache, PHP, and others. MacOS is secure against malware and attacks due to its Unix-based architecture. However, Macs are costly in terms of hardware, which can increase the overall cost of hosting. | Linux is a highly customizable and open-source operating system. Popular Linux distributions include Ubuntu, CentOS, and Red Hat, which cater to different use cases. Linux is lightweight and performs exceptionally well under heavy server loads. However, some Linux software or systems may not run as easily as they do on Windows or macOS. | Windows OS is widely utilized for hosting web applications that require.NET or ASP.NET environments. It includes Internet Information Services (IIS), a powerful platform for hosting websites and applications. The Windows OS is essential for web applications that are written in.NET or ASP.NET. It has a more familiar GUI, making it easier for those unfamiliar with command-line interfaces. However, Windows has been a frequent target for malware, although recent versions have improved security significantly. | Mobile devices are not commonly used for web hosting, but with the advancements in hardware, mobile devices can theoretically host lightweight applications. Mobile devices typically function as clients rather than servers. Mobile devices can be utilized in edge computing scenarios to provide localized applications that interact with cloud-based systems. However, mobile devices lack the processing capability and memory capacity of dedicated servers. They are not suitable for handling large-scale applications or traffic loads. |
| **Client Side** | Mac hardware is typically more expensive than other systems, so development teams will need to invest in Apple hardware for testing and development. Apple's design guidelines for macOS require adaptation of UI/UX principles to meet Apple's specifications. | Linux is more affordable since it utilizes many development tools that are open-source and free. Linux has a wide variety of distributions (Ubuntu, Fedora, etc.), which may require more extensive testing across different environments. | Windows is typically more affordable than Mac hardware, though Windows-specific tools or frameworks may have additional costs. Development for Windows typically involves following Microsoft's guidelines for UI/UX and ensuring compatibility with older versions of Windows, such as Windows 7 or Windows 10. | For iOS development, Mac hardware is necessary, while Android development can be done on various systems. However, mobile devices will be necessary for testing. Mobile development often requires strict UI/UX regulations, such as Apple's Human Interface Guidelines or Google's Material Design. |
| **Development Tools** | Proficiency in Swift and Objective-C programming is needed for macOS applications. | C, C++, C#, Python, and Java programming languages are frequently used in Linux environments. | Proficiency in C, C#, Python, Java, .NET, and Windows API is essential for Windows applications. | iOS development requires proficiency in Swift or Objective-C, while Android development requires Java or Kotlin languages. |

## Recommendations

1. **Operating Platform**: I would recommend using Windows to convert Draw It or Lost It! to a web-based platform.
2. **Operating Systems Architectures**: Windows utilizes several components that support the operating system. The Windows platform has several layers and types of architectures, such as hardware architecture, kernel architecture, and user mode architecture. The Windows platform architecture is designed to be flexible, providing various hardware configurations and user environments. Its hybrid kernel design enhances performance and stability, while its user mode and security model ensure a secure and secure system for running applications.
3. **Storage Management**: Microsoft Azure would be a great option for Windows storage. The cloud-based storage allows numerous scalability, security, and recovery options, and would cost much less than physical storage across many servers.
4. **Memory Management**: Windows employs virtual memory to allow applications to use more memory than physically available (RAM) by using a portion of the hard drive as "virtual" memory. When Draw It or Lose It! requires more memory than is available in physical RAM, Windows will store less-used data in a paging file on the disk, freeing up RAM for immediate needs, such as drawing operations. This allows the application to operate smoothly even when memory is low, although virtual memory is slower than physical RAM.
5. **Distributed Systems and Networks**: Distributed systems require multiple components (servers, databases, services, etc.) that operate on different devices but work together to provide a unique experience. For Draw It or Lose It!, this could mean having different platforms (e.g., mobile apps, web apps, desktops) interact with a central system. The speed of the network impacts the user experience. In a multiplayer game, it is essential to minimize latency to ensure drawings and responses are updated in real-time across multiple platforms. In a multiplayer game, it is essential that all platforms remain synchronized in real-time. If one platform has a delay or loses connection, it can disrupt the gameplay for all users. Event-driven architectures provide a way to maintain real-time sync.
6. **Security**: The protection of user information across various platforms involves implementing multiple layers of security measures that protect data both at home and in transit. Windows OS offers BitLocker, a full disk encryption feature that protects data stored on a device by encrypting the hard drive. Windows OS uses Windows Defender, a built-in antivirus and antimalware solution, to provide real-time protection against malware, viruses, and other threats. Finally, Windows supports Azure Backup for cloud-based backup solutions, ensuring regular backups of critical user data, stored in secure, encrypted formats.