

# 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 | 11/24/2024 | Khari Lyles-Shaw | Updated the Executive Summary, Design Constraints, and Domain Model sections of the template. |

**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 Gaming Room, a client of Creative Technology Solutions (CTS), has requested the development of a web-based version of their popular game "Draw It or Lose It." The game is loosely based on the 1980s television game "Win, Lose or Draw," where teams compete to guess what is being drawn. In the web-based version, the application will render images from a library of stock drawings as clues, and teams will have to guess the puzzle before the time expires.

To facilitate the development of the web-based game application, CTS has asked me to prepare a software design document and begin the development process.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

The primary design constraints for developing the game application in a web-based distributed environment are:

1. Cross-platform compatibility: The application must be accessible and functional across different operating systems (Mac, Linux, Windows) and mobile devices, ensuring a seamless user experience.
2. Scalability and performance: The application must be able to handle multiple concurrent users, teams, and games without compromising performance or stability.
3. Real-time interaction: The game requires real-time communication and synchronization between players, teams, and the game server, which must be addressed in the design.
4. Unique identifiers: The requirement to have unique identifiers for each instance of a game, team, or player poses a challenge in a distributed environment, as the identifiers must be consistently generated and managed across the system.
5. Singleton pattern: The need to have only one instance of the game in memory at any given time necessitates the use of a Singleton design pattern, which has implications on the overall application architecture.

These design constraints have a significant impact on the application development, requiring careful consideration of the system architecture, communication protocols, data management, and overall software design to ensure the successful implementation of the web-based version of "Draw It or Lose It."

## [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 the domain model for the "The Gaming Room" application. The key classes and their relationships are as follows:

1. Entity Class:
   * This is an abstract base class that encapsulates common attributes (id, name) and behaviors (getId(), getName(), toString()) shared by other domain entities.
   * This promotes code reuse and follows the principle of abstraction in object-oriented programming.
2. Player Class:
   * Represents an individual player in the game.
   * It inherits from the Entity class, leveraging the common attributes and methods.
   * Each Player has a unique id and name.
3. Team Class:
   * Represents a team of players in the game.
   * It also inherits from the Entity class, sharing the common attributes and methods.
   * Each Team has a unique id and name.
   * A Team can have multiple Players, stored in a List<Player>.
   * The Team class provides methods to add players (addPlayer()) and get a string representation (toString()).
4. Game Class:
   * Represents a single game instance.
   * It has a unique id and name.
   * A Game can have multiple Teams, stored in a List<Team>.
   * The Game class provides methods to add teams (addTeam()) and get a string representation (toString()).
5. GameService Class:
   * This class acts as a service provider for the Game-related functionalities.
   * It maintains a list of all the games (games: List<Game>), and provides methods to manage the games, such as addGame(), getGame(), and getGameCount().
   * It also provides methods to get the next available player and team IDs (getNextPlayerId(), getNextTeamId()).
   * The GameService class follows the Singleton design pattern, as indicated by the getInstance() method, ensuring only one instance of the class exists.

The overall design demonstrates the use of object-oriented principles, such as abstraction, inheritance, encapsulation, modularity, and the Singleton pattern. These principles enhance the flexibility, maintainability, and scalability of the application, addressing the key requirements of managing players, teams, and games.

**"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** | Mac can host web applications using macOS Server, but it has limited scalability and performance compared to other platforms. Licensing costs may be lower than Windows but still exist for server software. | Linux is highly efficient for server hosting, offering robust performance and scalability for thousands of users. Most distributions are open-source, minimizing licensing costs. | Windows Server provides strong support for hosting applications and is widely used in enterprise environments. However, licensing costs can be significant. | Mobile platforms like iOS and Android can host web applications but require specific configurations. Cloud-based solutions can help scale effectively. |
| **Client Side** | Developing for Mac requires expertise in Swift or Objective-C for native applications. Compatibility with web browsers is generally good, but ensuring a responsive design is crucial. | Linux development often utilizes languages like Python and Java. Ensuring compatibility with various web browsers can be challenging due to fragmentation. | Windows supports a wide range of development tools and languages, making it easy to develop responsive web applications. However, time and cost may increase due to the need for testing across different versions. | Developing for mobile requires knowledge of platform-specific languages (Swift for iOS, Kotlin for Android) and frameworks that ensure compatibility across devices. |
| **Development Tools** | Common IDEs include Xcode and Visual Studio Code, which may have associated licensing costs. Mac development may require a team familiar with Apple ecosystems. | Popular IDEs include Eclipse and Visual Studio Code, which are generally free. Linux development may require a diverse team familiar with various distributions. | IDEs like Visual Studio are powerful for Windows development but can incur licensing fees. A team skilled in Windows technologies may be necessary. | Mobile development tools like Android Studio and Xcode are essential. Knowledge in cross-platform frameworks (such as React Native) can help minimize development costs. |

## 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**: Linux is the recommended operating platform for expanding Draw It or Lose It into diverse computing environments. Its open-source nature eliminates licensing costs, making it an economical choice for long-term scalability. Furthermore, Linux offers exceptional performance and reliability, which are essential for supporting a real-time multiplayer game. Its compatibility with modern cloud environments ensures that the platform can adapt to increasing user demands while maintaining consistent service quality.
2. **Operating Systems Architectures**: The Linux operating system employs a modular architecture, with a monolithic kernel that integrates critical components such as memory management, process scheduling, and hardware communication. This design promotes efficient performance under high user loads, making it particularly suited for hosting multiplayer games. Additionally, Linux supports microservices architecture, allowing game components to be deployed and updated independently, which enhances flexibility and minimizes downtime during updates.
3. **Storage Management**: A cloud-based storage system, such as AWS S3 or Google Cloud Storage, is highly suitable for managing the data associated with Draw It or Lose It. These systems provide secure, scalable, and reliable storage solutions capable of handling the game’s data needs. Features such as version control, automatic backups, and easy integration with distributed environments ensure that player information, game configurations, and team data are consistently accessible and protected against data loss.
4. **Memory Management**: The recommended Linux platform employs advanced memory management techniques, including paging, caching, and virtual memory, to optimize resource allocation. These techniques dynamically distribute system resources, ensuring efficient performance even when multiple game sessions occur simultaneously. The platform’s ability to prioritize real-time processes ensures smooth gameplay and prevents lag, a critical factor for maintaining a high-quality user experience in a competitive, fast-paced environment.
5. **Distributed Systems and Networks**: To enable communication across various platforms, Draw It or Lose It can leverage distributed systems supported by RESTful APIs or WebSocket protocols. These technologies facilitate real-time synchronization between devices, regardless of their operating system. To ensure consistent performance and minimize disruptions, a robust network infrastructure should include load balancers and failover mechanisms to manage connectivity and mitigate potential outages. This setup ensures that players can seamlessly interact across platforms, enhancing the overall gaming experience.
6. **Security**: Security is a top priority for protecting user data across platforms. The system should employ end-to-end encryption, such as TLS, to secure data transmission. Multi-factor authentication (MFA) adds an additional layer of security to user accounts, while role-based access control (RBAC) ensures that only authorized personnel have access to sensitive data. Regular security audits, penetration testing, and compliance with data protection regulations, such as GDPR and CCPA, further strengthen the platform’s security framework and build user trust.