

<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 | <02/25/24> | <Jacob Rethmeier> | <All changes included: design constraints, system architecture, executive summary, domain model, development requirements and included all 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 objective of the Gaming Room is to create a cross-platform web game inspired by the existing game "Draw It or Lose It," currently exclusive to Android. The game is designed for multiple teams, each composed of several participants, engaging in four rounds, each lasting a minute. In each round, teams take turns guessing the content of a randomly selected image from a library. If the team fails to guess within the time limit, opposing team members have an opportunity to answer within a 15-second timeframe.

## [Design Constraints](#_2et92p0)

* A game must be able to have multiple teams playing
* Each team must consists of multiple players
* Both game and team names must be unique and if the name is taken notify the user
* Only one instance of the game at a time(unique identifiers for each game/team/player)

## [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 ProgramDriver class serves as an entry point and utilizes the SingletonTester class, which includes a method for testing singleton behavior. A foundational Entity class, encapsulates common attributes and methods for various entities in the system, such as an identifier and a name. Entity is a superclass associated with Game, Team, and Player classes through a dependency relationship. The system employs a singleton design pattern with the GameService class, responsible for managing a list of Game instances. Each Game instance, in turn, maintains a list of Team instances, and each Team comprises multiple Player instances. The relationships between these classes are depicted through one-to-many connections. The overall structure establishes a framework for managing games, teams, and players within the system.

**"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** | macOS Server provides some server capabilities but is less common for web hosting compared to Linux or Windows. Integration with the macOS ecosystem may provide ease of use for developers familiar with Apple products. However, it has limited scalability compared to Linux or Windows servers. Hosting websites using macOS Server is possible but less common, typically involving tools like Apache or Nginx. Licensing costs are associated with Apple products. | Linux offers robust server capabilities with a wide array of server distributions like Ubuntu Server, CentOS, etc. Its open-source nature allows for cost-effectiveness and customization, with strong support for server-side technologies like Apache, NGINX, Node.js, etc. Hosting methods typically involve using Apache or NGINX to host websites. Licensing costs are minimal, typically only associated with support services if needed. | Windows Server is widely used in enterprise environments for hosting web applications. Its user-friendly interface and good support for Microsoft technologies like IIS, .NET, etc., make it popular. However, it comes with higher licensing costs compared to Linux. Hosting websites on Windows Server involves using IIS, which provides extensive features. Licensing costs vary based on the edition and usage of Windows Server. | Developing server-side components for mobile devices involves implementing APIs, databases, and authentication mechanisms optimized for mobile use cases. These components must be scalable, efficient, and secure to handle requests from thousands of mobile devices simultaneously. Considerations for intermittent connectivity and variable bandwidth are essential. Overall, it requires a comprehensive understanding of mobile app architecture and backend technologies. |
| **Client Side** | Mac users access web applications through modern, responsive HTML interfaces running inside web browsers. Development for Mac desktops may require familiarity with macOS-specific development tools and frameworks. For mobile platforms (iOS), applications can be built using Swift or Objective-C for native development, or frameworks like Flutter or React Native for cross-platform development. | For desktop clients, Linux users access web applications through modern, responsive HTML interfaces running inside web browsers like Chrome, Firefox, etc. Development for Linux desktops may require expertise in web development technologies. On mobile platforms, applications can be built using frameworks like Flutter, React Native, or by developing separate native applications. | Windows users access web applications through modern, responsive HTML interfaces running inside web browsers like Chrome, Firefox, Edge, etc. Development for Windows desktops may require expertise in web development technologies. On mobile platforms, applications can be built using frameworks like Xamarin for native development, or Flutter, React Native for cross-platform development. | Developing client-side components for mobile devices involves creating intuitive, responsive user interfaces using modern web technologies like HTML5, CSS3, and JavaScript frameworks. Ensuring cross-platform compatibility and adherence to mobile design principles are essential for enhancing user experience. Integration with device features and performance optimization are also crucial for delivering compelling mobile applications. |
| **Development Tools** | Programming Languages: Swift, Objective-C, Java, Kotlin, C++  Tools: IDEs like Xcode, JetBrains IDEs, and development frameworks like CocoaPods, SwiftUI, etc. | Programming Languages: C++, Java, Python, JavaScript (Node.js), HTML5, CSS3  Tools: IDEs like Visual Studio Code, Eclipse, JetBrains IDEs, and development frameworks like Node.js, React, Angular, etc. | Programming Languages: C#, C++, Java, JavaScript (Node.js), Python  Tools: IDEs like Visual Studio, Visual Studio Code, JetBrains IDEs, and development frameworks like .NET, Xamarin, React, etc. | Programming Languages:  Swift (iOS)  Objective-C (iOS)  Java (Android)  Kotlin (Android)  Tools:  Xcode (iOS)  Android Studio (Android)  Flutter  React Native  CocoaPods (iOS)  Android Emulator |

## 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:**
   * Windows is recommended for its extensive support for programming languages and tools, providing a rich ecosystem for game development. Its user-friendly interface and moderate cost make it an attractive choice for the project. However, consider evaluating other operating platforms like Linux and macOS for their compatibility and scalability with the project's requirements.
2. **Operating Systems Architectures:**
   * Windows operates on a hybrid kernel architecture, combining aspects of monolithic and microkernel designs. This architecture supports efficient process and memory management, critical for gaming applications like Draw It or Lose It. Additionally, Windows provides a robust graphical user interface (GUI) and system APIs, facilitating development across various computing environments.
3. **Storage Management:**
   * While Windows offers built-in storage management features like Storage Sense, leveraging cloud-based storage solutions like Amazon S3 can enhance scalability and reliability. Integrating Amazon S3 with the Draw It or Lose It application ensures efficient data storage, backup, and accessibility across different platforms.
4. **Memory Management:**
   * Windows' virtual memory system dynamically manages memory allocation, allowing Draw It or Lose It to utilize both RAM and storage resources effectively. This ensures optimal performance and responsiveness, particularly during gameplay sessions with loading images. Windows Task Manager provides real-time monitoring of memory usage, enabling users to optimize system resources.
5. **Distributed Systems and Networks:**
   * Implementing a distributed system architecture enables Draw It or Lose It to communicate seamlessly across various platforms. By designing the software to be platform-agnostic and utilizing network protocols like TCP/IP, the game can interact with clients on different devices. However, considerations should be made for connectivity issues, outages, and network latency to ensure a smooth gaming experience.
6. **Security:**
   * Security measures are paramount to safeguard user information in Draw It or Lose It. Implementing a multi-layered strategy with robust user authentication mechanisms, secure communication protocols (such as HTTPS), and data encryption ensures confidentiality and integrity. Utilizing secure storage solutions and device authentication further enhances security across multiple platforms, mitigating risks of unauthorized access and data breaches.