

<Name-of-Software-Application>

# **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 | 11/15/23 | Jonathan Plummer | <Brief description of changes in this revision> |

**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)

**Introduction and Solution Overview:**

The Gaming Room's web-based game "Draw It or Lose It" presents a design challenge of creating a collaborative and competitive puzzle-solving environment, inspired by the 1980s TV game "Win, Lose or Draw." The game's core mechanics revolve around teams guessing puzzles based on rendered images, necessitating a robust system to manage multiple teams, players, and unique game instances. Our proposed solution is a Java-based application, leveraging object-oriented principles for modularity and scalability. Key components include the GameService class, implementing a singleton pattern for singular game instance management; the Game class for handling game instances with unique identifiers and name uniqueness; the Team class for managing teams and players; and the Player class, ensuring unique player identification within teams.

**Critical Considerations and Next Steps:**

Addressing the client's requirements, our design ensures name uniqueness for games and teams, crucial for user experience and data integrity, and accommodates future scalability and feature enhancements. The current focus is on backend logic, and we recommend parallel frontend development for a cohesive user interface. The next steps involve finalizing the requirement interpretations, especially regarding the single-game-instance constraint, followed by rigorous development and testing phases. This phase ensures all system components interact seamlessly, laying a robust foundation for "Draw It or Lose It," setting the stage for a successful launch and potential future expansions.

## 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)

Developing "Draw It or Lose It" in a web-based distributed environment imposes several design constraints. Firstly, ensuring real-time interaction and seamless rendering of images demands a robust backend and efficient data transmission protocols. The uniqueness of game and team names necessitates a carefully designed data management system to prevent conflicts and maintain consistency. Additionally, the singleton pattern used in GameService must be managed to prevent any potential memory leaks or performance bottlenecks, particularly crucial in a distributed environment. These constraints will influence our choice of technologies, data handling mechanisms, and overall application performance optimization strategies.

## [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 GameService class is a singleton, which is evident from the self-referencing -service: GameService attribute and the +getInstance(): GameService method. The singleton pattern ensures that only one instance of this class can exist in the system, meeting the requirement that only one instance of the game can exist at any given time.

The Game class is associated with GameService through a one-to-many relationship, as indicated by the "0..\*" next to the GameService class. This means an instance of GameService can manage multiple games.

Each Game contains a list of Team instances, represented by the "-teams: List<Team>" attribute, suggesting a composition relationship. Composition is a form of aggregation that implies that Team objects are dependent on Game for their lifecycle.

The Team class has an aggregation relationship with the Player class, as shown by the "-players: List<Player>" attribute and the "0..\*" multiplicity, indicating that a Team can have multiple Player objects, but these Player instances can exist independently of the Team.

The Entity class serves as a base class for Game, Team, and Player, demonstrated by the generalization arrows pointing from these classes to Entity. This indicates that they inherit the ID and name attributes, as well as common behaviors, from Entity. This showcases the use of inheritance and encapsulation, with Entity encapsulating the common properties and providing a reusable interface for its subclasses.

The ProgramDriver and SingletonTester classes are utility classes. ProgramDriver contains the main method, serving as the entry point of the application. SingletonTester seems to be a testing utility to verify the singleton behavior of GameService. The use relationship (dependency) from SingletonTester to GameService indicates that SingletonTester uses GameService but does not manage its lifecycle.

Object-Oriented Principles Demonstrated:

Encapsulation: The diagram suggests encapsulation through the use of private attributes (though not explicitly marked in the diagram) and public methods in the classes.

Inheritance: The Entity class is inherited by Game, Team, and Player, promoting code reuse and a clear hierarchy.

Polymorphism: While not explicitly shown in this diagram, polymorphism would be enabled by the inheritance from the Entity class, allowing objects of Game, Team, and Player to be treated as Entity objects where appropriate.

Abstraction: The Entity class provides an abstraction layer for the shared properties and behaviors of Game, Team, and Player.

Fulfillment of Software Requirements:

The singleton pattern used in GameService ensures that the software requirement of having only one game instance in memory is met.

The unique identifiers provided by the Entity class fulfill the requirement for unique entities within the system.

The aggregation and composition relationships ensure that teams and players are managed correctly within games, adhering to the requirement that games have multiple teams and teams have multiple players.

The uniqueness of game and team names can be enforced by the logic within the addGame and addTeam methods, which is implied but would need to be confirmed in the actual method implementations.

This diagram efficiently communicates the foundational class structure and relationships for "The Gaming Room" application, aligning with the specified software requirements.

**"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 offers characteristics such as a Unix-based operating system, robust terminal support, and compatibility with a wide range of development tools. Its advantages include a strong developer community, support for containerization with Docker, and reliable server performance. However, Mac may have weaknesses in terms of hardware costs and limited scalability for large-scale enterprise applications. | Linux, as a server-side platform, boasts characteristics like open-source availability, security, and stability. Its advantages include cost-effectiveness, customizability, and support for a wide array of programming languages and technologies. Linux is known for its robust server performance and scalability. However, it may have weaknesses in terms of desktop GUI tools, requiring more familiarity with the command line for server administration. | Windows, as a server-side platform, provides characteristics such as a user-friendly interface, extensive developer tools, and compatibility with Microsoft technologies. Advantages include seamless integration with other Windows-based systems, support for ASP.NET and C#, and a strong presence in enterprise environments. However, Windows servers may have weaknesses related to licensing costs, potential security vulnerabilities, and a steeper learning curve for command-line administration. | Mobile devices, when used for server-side development, have characteristics like portability and diverse hardware capabilities. Advantages include the ability to build cloud-based services, leverage mobile-specific features like push notifications, and reach a broad user base. However, weaknesses encompass limited computational power, potential connectivity issues, and the need for optimized server responses to accommodate mobile network conditions. Developers must consider these factors when developing server-side solutions for mobile devices to ensure a seamless experience. |
| **Client Side** | Developing software for Mac clients involves cost considerations such as the need for Mac hardware and developer licenses. Time may be required for testing and optimization on Mac-specific hardware and OS versions. Expertise in macOS, Swift programming, and Apple's development tools like Xcode is crucial for Mac application development. | Linux development typically has lower costs due to open-source tools, but consideration must be given to compatibility across various Linux distributions. Time efficiency can be achieved with the availability of open-source libraries, while expertise in Linux system administration and programming languages commonly used on Linux is essential. | Developing for Windows may incur costs related to licensing and testing on different Windows versions. Time may be extended due to the diversity of Windows environments. Expertise in C#, .NET, and familiarity with Windows-specific APIs and tools like Visual Studio is vital for Windows application development. | Developing for mobile devices can be costly due to the need for multiple testing devices and licensing fees for app stores. It can also be time-consuming, with platform-specific development and rigorous testing on various devices. Expertise in platform-specific languages (Swift, Java/Kotlin), mobile app design, and knowledge of development environments like Xcode and Android Studio is essential for mobile app development. |
| **Development Tools** | Developing software for Mac involves using programming languages like Swift and Objective-C within the Xcode IDE. Xcode provides a comprehensive development environment specifically tailored to macOS and iOS app development. It offers features for designing user interfaces, writing code, debugging, and testing, making it the go-to choice for Mac application development. However, one consideration is that Mac development is exclusive to the Apple ecosystem, limiting the potential audience to macOS and iOS users. | Linux software development is characterized by its openness and flexibility, allowing developers to choose from a variety of programming languages such as C, C++, Python, and Java. Open-source IDEs like Visual Studio Code, CLion, or Eclipse provide powerful development tools. Linux offers a diverse ecosystem and is well-suited for server-side and cross-platform development. However, it may require more manual configuration and compatibility testing due to the multitude of Linux distributions and hardware variations. | Windows development primarily relies on languages like C#, C++, and .NET, with Visual Studio serving as the flagship IDE. Visual Studio is a comprehensive development environment that streamlines Windows application development, offering features like code editing, debugging, and extensive libraries. Windows applications have broad reach, covering desktop, web, and mobile platforms. However, it's essential to consider licensing costs and the Windows-centric nature of this development environment. | For mobile devices, iOS development on Mac involves Swift and Xcode for creating apps exclusively for Apple devices. On the Android side, development typically occurs on various platforms but is rooted in Java and Kotlin, with Android Studio as the primary IDE. Both platforms offer extensive toolsets for creating mobile applications, but developers must consider the unique design guidelines and app store requirements for each ecosystem. Cross-platform frameworks like Flutter and React Native can also be advantageous for targeting both iOS and Android simultaneously, reducing development time and 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**: To expand "Draw It or Lose It" to other computing environments and ensure cross-platform compatibility, I recommend using a Linux-based server as the hosting platform. Linux is known for its versatility and wide-ranging compatibility with various computing environments. By deploying the game on a Linux server, you can easily accommodate users on Windows, macOS, and even mobile devices through web-based access. Linux servers are commonly used for web hosting and can support web applications, making it an ideal choice for expanding the game's reach while maintaining cost-efficiency and flexibility. This approach allows for seamless access from diverse operating systems, enabling The Gaming Room to tap into a broader user base.
2. **Operating Systems Architectures**: The chosen operating platform architecture for expanding "Draw It or Lose It" to other computing environments is based on a Linux server environment. Users on various computing environments, including Windows, macOS, and mobile devices, will access the game through web browsers. A Linux operating system running on robust server hardware will host the game's backend logic, and a web server software like Apache or Nginx will handle incoming HTTP requests. Data will be stored in a relational database management system, ensuring data integrity and efficient retrieval. The game's application logic will be implemented using server-side programming languages like Java, Python, or Node.js, and communication between the client and server will occur over standard HTTP or HTTPS protocols. This architecture provides scalability, security, and platform-agnostic access, meeting the client's requirements for cross-platform compatibility and future growth.
3. **Storage Management**: The recommended storage management system to be used with the Linux-based operating platform for "Draw It or Lose It" is MySQL. MySQL is a robust, open-source relational database management system known for its scalability, performance, and reliability. It can efficiently handle the game's data storage needs, including player information, game state, and puzzle data.
4. **Memory Management**: The recommended operating platform, Linux, employs advanced memory management techniques to enhance the performance and efficiency of the "Draw It or Lose It" software. Linux utilizes virtual memory to allocate memory space for processes, ensuring that the game application can access the memory it needs without concerns about physical memory limitations.
5. **Distributed Systems and Networks**: achieving cross-platform communication in "Draw It or Lose It" involves implementing a client-server architecture, handling connectivity challenges, and ensuring data synchronization. A robust network infrastructure and scalable server resources are vital to deliver a seamless and enjoyable gaming experience across various platforms.
6. **Security**: To protect user information on and between various platforms in "Draw It or Lose It," a comprehensive security strategy should be adopted. This strategy includes implementing robust encryption protocols for data transmission, enforcing strong authentication and authorization mechanisms, securing user data in storage through encryption and patch management, adhering to secure coding practices, and establishing continuous monitoring and logging systems. Additionally, user privacy controls, compliance with data protection regulations, and user education on security best practices should be integrated into the application. The selected operating platform should align with these security requirements, leveraging its built-in security features while also considering third-party security tools and practices to ensure the utmost protection of user information and maintain a secure gaming environment.