

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.3

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View**](#_heading=h.4d34og8) **4**

[**Domain Model**](#_heading=h.2s8eyo1) **4**

**Prog. Principals****5**

[**Evaluation/Server-side**](#_heading=h.17dp8vu) **6**

[**Evaluation/Client-side**](#_heading=h.17dp8vu) **7**

[**Evaluation/Dev.-tools**](#_heading=h.17dp8vu) **8**

[**Recommendations**](#_heading=h.26in1rg) **9**

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/16/2023 | Joseph Dengler | Project One requirements implemented |
| 1.1 | 10/01/2023 | Joseph Dengler | Project Two requirements implemented; evaluation and development requirements |
| 1.3 | 10/10/2023 | Joseph Dengler | Project Three requirements implemented; Analyzed the characteristics of and techniques specific to various systems architectures and made a recommendation to The Gaming Room. |

## [Executive Summary](#_heading=h.35nkun2)

Creative Technology Solutions (CTS) is thrilled to embark on a partnership with The Gaming Room, a client looking to expand their popular Android game, "Draw It or Lose It," into a web-based application, catering to multiple platforms. In response to your requirements, we are excited to present a comprehensive software design solution that will streamline the development of this exciting gaming venture.

## Requirements

Cross-Platform Compatibility Requirement:

Business Requirement: Ensure that the game can be accessed by players on various web browsers and operating systems, including desktop and mobile devices.

Technical Requirement: Extensive testing and potential platform-specific optimizations to guarantee a seamless user experience across different devices.

Network Latency Requirement:

Business Requirement: Deliver a real-time gameplay experience with minimal delays caused by network latency.

Technical Requirement: Implementation of latency compensation mechanisms, including client-side prediction and server reconciliation, to mitigate the impact of network delays on gameplay.

Scalability Requirement:

Business Requirement: Accommodate a large number of concurrent players and game instances to ensure a scalable and enjoyable gaming experience.

Technical Requirement: Development of a robust server infrastructure capable of handling increased player loads, potentially utilizing cloud-based services for scalability.

## [Design Constraints](#_heading=h.1ksv4uv)

Cross-Platform Compatibility: The game must run smoothly on various web browsers and operating systems, including desktop and mobile devices. This constraint necessitates extensive testing and potentially platform-specific optimizations to ensure a consistent user experience.

Implication: Developers need to prioritize compatibility and may need to employ responsive design techniques to adapt the game's interface to different screen sizes and resolutions.

Network Latency: In a web-based distributed environment, network latency can affect the real-time gameplay experience, causing delays in rendering, player input, and communication between game clients and the server.

Implication: Developers must implement latency compensation mechanisms to minimize the impact of network delays, such as client-side prediction and server reconciliation.

Scalability: The game must accommodate a potentially large number of concurrent players and game instances. Scalability constraints require the use of scalable server architecture and databases.

Implication: Development efforts should focus on building a robust server infrastructure that can handle increased player loads. This may involve utilizing cloud-based services for scalability.

## [System Architecture View](#_heading=h.44sinio)

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](#_heading=h.2jxsxqh)

ProgramDriver Class class is the entry point of the program and contains a main() method. It is not directly related to the other classes in the diagram but uses the SingletonTester class.

The SingletonTester class contains a testSingleton() method. It is used by the ProgramDriver class, denoted by the <<uses>> relationship. However, its role in the overall application is not clear from the diagram.

The Entity class is a base class with common attributes and behaviors for other classes. It includes attributes id (of type long) and name (of type String). It provides getter methods for these attributes and a toString() method.

The GameService class appears to manage game-related functionality. It has attributes such as games, nextGameId, nextPlayerId, nextTeamId, and service. It seems to implement the Singleton design pattern, as it includes a getInstance() method. It also provides methods for adding games, retrieving games by ID or name, and managing player and team IDs.

The Game class represents a game instance and has an association with the Team class. It has methods for adding teams and generating a string representation of the game.

The Team class represents a team within a game and is associated with the Player class. It has methods for adding players and generating a string representation.

The Player class represents an individual player and has a simple constructor and a toString() method.

**Class Relationships:**

There is an inheritance relationship between the GameService, Game, Team, and Player classes with the Entity class, indicating that they inherit common attributes and behaviors from Entity.

The GameService class is connected to the Game class with a "0...\*" multiplicity, indicating that a GameService can manage multiple games.

The Game class is connected to the Team class with a "0...\*" multiplicity, indicating that a game can have multiple teams.

The Team class is connected to the Player class with a "0...\*" multiplicity, indicating that a team can have multiple players.

**Object-Oriented Programming Principles:**

Inheritance: The use of the Entity class as a base class for GameService, Game, Team, and Player demonstrates the inheritance principle. It allows these classes to inherit common attributes and behaviors, promoting code reusability.

Association: The associations between GameService and Game, Game and Team, and Team and Player represent relationships between objects, facilitating the organization of game-related data.

Singleton Design Pattern: The GameService class implements the Singleton design pattern by providing a single point of access to its instance via the getInstance() method. This ensures that only one instance of GameService exists, which aligns with the requirement for only one instance of the game to exist in memory.

**"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](#_heading=h.z337ya)

By integrating the outlined development requirements and evaluating the characteristics, advantages, and weaknesses of each operating platform and mobile devices, it's clear that each option comes with its own set of considerations. The choice of platform should align with The Gaming Room's budget, scalability needs, and the expertise of the development team. It's essential to strike a balance between compatibility, cost-effectiveness, and user experience to create a successful and inclusive gaming application.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac operating systems are primarily designed for personal computing and are not commonly used as server platforms. While it's possible to host a web application on macOS, it might not be the optimal choice in terms of scalability and cost-efficiency. Licensing costs could be relatively high for Mac server configurations. | Linux offers robust server-based deployment options, supporting web hosting. It is renowned for its stability and security, making it an excellent choice for hosting web applications. Licensing costs for Linux server operating systems are typically lower compared to proprietary alternatives, aligning with cost-effectiveness. | Windows Server operating systems provide extensive support for hosting web applications. They offer user-friendly interfaces and a wide range of compatible software. However, licensing costs for Windows Server OS can be significant, impacting the overall project budget. | Mobile devices, including Android and iOS platforms, require a different approach. The server-side logic will not be hosted on the devices themselves but on dedicated servers or cloud platforms. Both platforms can communicate seamlessly with the server-side web application, allowing cross-platform gameplay experiences. |
| **Client Side** | Software Development Considerations:  Developing for Mac requires considerations for the macOS ecosystem. While macOS supports web technologies, the challenge lies in ensuring seamless compatibility with various browsers like Safari, Chrome, and Firefox. Developers need expertise in responsive design to adapt the application to different screen sizes and resolutions commonly found in Mac devices.  Cost, Time, and Expertise:  The development team must invest time in thorough testing across different Mac devices and browsers. Expertise in macOS-specific design guidelines and frameworks, along with compatibility testing, will impact development costs. | Software Development Considerations:  Linux users utilize a variety of browsers, including Firefox, Chrome, and others. Ensuring compatibility across these browsers is vital. Additionally, Linux desktop environments and configurations vary widely, requiring careful consideration to make the application universally accessible.  Cost, Time, and Expertise:  Linux development might require additional time for testing due to the diverse ecosystem. While Linux development tools are typically open-source and cost-effective, expertise in Linux-specific design considerations is necessary. | Software Development Considerations:  Windows users primarily rely on browsers like Chrome, Firefox, and Edge. Compatibility across these browsers is crucial. Moreover, developers need to consider different versions of Windows and potential variations in user settings that might affect web application performance.  Cost, Time, and Expertise:  Developing for Windows demands expertise in Windows-specific user interface guidelines and responsive design principles. Thorough testing across Windows versions and browsers will impact development costs and time. | Software Development Considerations:  For mobile devices, developers must focus on creating a responsive web interface compatible with various mobile browsers (Safari for iOS and Chrome for Android). Touchscreen interactions, varying screen sizes, and resolutions need to be accounted for to ensure an optimal user experience.  Cost, Time, and Expertise:  Mobile development requires expertise in mobile-specific design patterns and responsive web design. Extensive testing on different devices and screen sizes is necessary, impacting development costs and time. |
| **Development Tools** | Relevant Programming Languages and Tools:  For Mac, developers commonly use Xcode, an integrated development environment (IDE) supporting Swift and Objective-C for iOS app development. Web technologies like HTML, CSS, and JavaScript can be developed using various text editors or IDEs compatible with macOS.  Impact on Development Team:  Xcode is a specialized tool for iOS app development. Developers focusing on Mac must have expertise in Swift, Objective-C, and macOS-specific development frameworks. | Relevant Programming Languages and Tools:  Linux supports a wide range of programming languages and development tools. Common choices include JavaScript frameworks like React.js and Node.js for web development. Text editors like Visual Studio Code are popular for web development on Linux.  Impact on Development Team:  Linux development tools are diverse and cost-effective. Developers can choose from various open-source tools, promoting flexibility and accessibility for development teams. | Relevant Programming Languages and Tools:  Windows development commonly involves tools like Visual Studio, which supports multiple programming languages including .NET, C++, and JavaScript. Web technologies can be developed using Visual Studio Code or other text editors compatible with Windows.  Impact on Development Team:  Visual Studio offers robust features, but it can be costly. Depending on the project requirements, developers might need expertise in .NET technologies, impacting team composition and budget. | Relevant Programming Languages and Tools:  For Android development, developers use Android Studio, supporting Java/Kotlin. For iOS, Xcode with Swift/Objective-C is used. Web technologies like HTML, CSS, and JavaScript are utilized for creating responsive web interfaces for mobile devices.  Impact on Development Team:  Developers need platform-specific expertise in Java/Kotlin for Android and Swift/Objective-C for iOS. Cross-platform compatibility requires thorough testing, potentially impacting development time and costs. |

## Recommendations

By adopting these recommendations, The Gaming Room can create a robust, scalable, and secure gaming platform for Draw It or Lose It, catering to diverse computing environments and ensuring a seamless gaming experience for players across different platforms.

1. **Operating Platform**: For the operating platform, we recommend utilizing a Linux-based server environment. Linux offers robust server-based deployment options, providing stability, security, and cost-effectiveness. Its open-source nature aligns well with budget considerations, allowing resources to be allocated more efficiently towards other aspects of the project. Additionally, Linux has extensive compatibility with various web technologies, ensuring a seamless experience for players across different platforms and browsers.
2. **Operating Systems Architectures**: The chosen Linux-based server environment can be implemented using a microservices architecture. Microservices facilitate scalability, allowing individual components of the game to be developed, deployed, and scaled independently. This architecture promotes modularity and flexibility, enabling easier integration with different platforms and technologies.
3. **Storage Management**: For storage management, we recommend utilizing a NoSQL database system such as MongoDB. NoSQL databases offer flexibility in handling unstructured or semi-structured data, making them suitable for storing diverse game-related information. MongoDB, in particular, provides horizontal scalability, ensuring that the database can grow seamlessly with the increasing player base. Its ability to handle large volumes of concurrent read and write operations aligns with the scalability requirements of Draw It or Lose It.
4. **Memory Management**: The recommended Linux-based server environment, coupled with the microservices architecture, allows efficient memory management for Draw It or Lose It. Each microservice can be individually optimized for memory usage, ensuring that resources are allocated optimally based on the specific requirements of different game components. Additionally, Linux provides robust memory management capabilities, including virtual memory and process isolation, further enhancing the stability and performance of the game.
5. **Distributed Systems and Networks**: To enable communication between various platforms, a RESTful API can be implemented using HTTP protocols. This API will serve as the bridge between different clients (web browsers, desktops, and mobile devices) and the server. Additionally, message queuing systems like RabbitMQ can be integrated to ensure asynchronous communication between microservices, enhancing fault tolerance and scalability. Load balancers can be employed to distribute incoming traffic across multiple server instances, ensuring high availability and efficient resource utilization.
6. **Security**: Security measures should be implemented at multiple levels to protect user information and ensure secure communication between platforms. HTTPS should be enforced to encrypt data transmitted between clients and the server, preventing unauthorized access during data exchange. Access control mechanisms should be established, restricting sensitive operations to authenticated and authorized users. Regular security audits and penetration testing should be conducted to identify and address potential vulnerabilities. Additionally, data at rest should be encrypted to safeguard stored information from unauthorized access.