

The Gaming Room

# **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 | 09/10/23 | April Nixon | Working game, team, and player classes  Add new class, Entity class |
| 1.1 | 09/11/23 | April Nixon | Added more evaluation information |
| 1.2 | 09/22/23 | April Nixon | Included more recommendations |
| 1.3 | 10/7/23 | April Nixon | Added Architecture Recommendations in paragraph form |

**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)

Our latest project involves assisting The Gaming Room in transforming their game, "Draw It or Lose It", into an online platform. Our main role will be to facilitate the creation of a gaming environment where numerous teams can participate, each consisting of several individual players. It is imperative that both the team names and player names are distinctive to avoid any confusion. Moreover, we need to integrate a feature that verifies the uniqueness of the names in real time to prevent duplication. The entire process must be streamlined and efficient to ensure that the platform can generate revenue successfully.

## [Design Constraints](#_2et92p0)

The primary challenges we anticipate during this project include adhering to the stipulated timeframe and budget. Our client, The Gaming Room, entrusts us with the responsibility of delivering a profitable application without delay. They envisage an application where only a single instance of the game can be active in the memory at any moment. Our task is to bring this vision to life, aligning the application's presentation seamlessly with The Gaming Room's expectations.

In addition, we must allocate time and resources judiciously to facilitate comprehensive testing of the software. This involves gathering and incorporating feedback from both the client and user experience testers to enhance the product's quality. These elements are essential and should be integrated within the existing budget and timeframe constraints.

## [System Architecture View](#_ilbxbyevv6b6)

1. **Logical Views**: These views focus on the core functions and services that the game system provides to the players. It's like looking at the "what" of the game, emphasizing its operational aspects and how it serves gamers.
2. **Process Views**: This perspective delves into the dynamic aspects of the game system. It explains how the game processes work during runtime and sheds light on the game's real-time behavior. In essence, it's about understanding "how" the game operates during gameplay.
3. **Physical Views**: Physical views are based on the viewpoints of the designers and developers. These views provide insights into how the game's components, such as servers, databases, and devices, are structured and interact with each other. It's about the tangible, technical aspects of the game's architecture.
4. **Development View**: The development view offers a view of the system from a programmer's standpoint. It's all about how the game is built, coded, and structured by the development team. It's especially important for game designers and those involved in creating the game's codebase.
5. **Scenarios**: Scenarios deal with the actual interactions within the game. They focus on how game objects interact with each other and how the game processes unfold during specific situations or gameplay scenarios. It's about understanding the practical, in-game experiences.

## [Domain Model](#_8h2ehzxfam4o)

The UML Diagram illustrated below outlines the Object-Oriented Programming (OOP) concepts that will be foundational in the development of this application. Our design entails the utilization of seven distinct classes to shape the application as envisioned.

At the core of the structure is the 'Entity' class, which serves as a parent class and establishes base attributes such as 'Id' and 'Name'. These attributes will be accessible through basic getter methods and will be inherited by the subsequent child classes: 'Game', 'Team', and 'Player'. Leveraging the inheritance feature in the OOP approach, we can streamline the development process, as the child classes readily inherit necessary variables from the parent class.

Diving deeper, the 'Game' class is responsible for housing a roster of teams, facilitating the addition of new team names to the roster. Meanwhile, the 'Team' class maintains a roster of active players and features a method to facilitate the enlistment of new players to the roster. The 'Player' class essentially facilitates the creation of individual player instances.

Furthermore, the 'GameService' class emerges as a pivotal component, interfacing with the other four classes following a singleton pattern, thereby ensuring a singular active game instance at any given moment. This class is also equipped with iterator patterns, fostering quick and efficient navigation through all the names, maintaining the uniqueness of the active instance.

To ensure the smooth functioning of our singleton class (GameService), a 'SingletonTester' class will be instituted, which will be operated by the 'ProgramDriver' class to conduct necessary testing. The 'ProgramDriver' class, housing the main method, takes the helm in steering the entire application, ensuring everything runs as per the designated blueprint.

**"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** | Utilizing Mac OS as a server platform isn't the most common choice, yet it remains a viable option. One of the significant hindrances of this route is the necessity for an expert who has a deep understanding of Mac OS operations to fully exploit its capabilities. On the flip side, one can benefit from a more customized server environment that is fine-tuned to maximize the performance of applications running on Mac OS. While the initial costs might be within a reasonable bracket, the requirement for dedicated personnel to oversee the server operations coupled with the OS licensing fees can potentially escalate the overall expenses. | Linux stands as the foremost choice for operating servers owing to its open-source nature, which makes the initial setup quite affordable and easy to manage. However, like the scenario with Mac OS, leveraging its full potential ideally calls for the expertise of a dedicated server administrator. An added advantage is the absence of licensing fees, which further contributes to its cost-effectiveness and popularity in the server management landscape. | Initiating a server with Windows OS entails a higher initial expenditure due to substantial licensing fees and setup costs. Despite this, Windows OS remains a favored option owing to its user-friendly nature and widespread recognition, which allows for a relatively straightforward administration process, often not necessitating dedicated personnel. Furthermore, the platform comes equipped with a plethora of integrated applications, enhancing the efficiency and functionality of the server, and ensuring an optimized utilization of the server's capabilities. | Utilizing a mobile device to host a server seems impractical, especially when measured against the robust capabilities of a dedicated computer system. The primary concerns here are not financial but are instead centered around the technical limitations inherent in mobile devices. These devices lack the substantial processing power and resources that dedicated computers have, making them significantly less suited for running servers efficiently. |
| **Client Side** | Mac requires a more hands-on approach as a client, necessitating someone well-versed in its operating system to guarantee compatibility with the respective coding. To prevent potential issues with cross-platform compatibility, it is recommended to maintain a simplistic approach to the application's structure and functionality. This strategy will help in avoiding any undue complications that might arise from the platform's particular requirements. | Just like with Mac, utilizing Linux as a platform necessitates having a specialist readily available due to its limited usage compared to other platforms. While this does not necessarily inflate the development cost, it would be prudent to streamline the team to those who are proficient with Linux to optimize both time and financial resources. This focused approach would ensure a smoother project trajectory without unnecessary expenses. | Utilizing Windows as the platform might potentially expedite the development process given its widespread use and familiarity among team members. The platform offers an abundance of tools that can assist in the development process, potentially shortening the time needed to create the application. It would be beneficial for the team to fully harness these available resources to streamline development and make the project more time efficient. | Mobile platforms are now a common medium for development and shouldn't cause significant delays in the project timeline. To guarantee a seamless operation across various operating systems, it is imperative that the team prioritizes regular testing throughout all phases of development, ensuring that the application functions optimally at every stage. |
| **Development Tools** | PHP programming language  JavaScript | PHP programming language | Java programming,  HTML/CSS  Netbeans | Android Studio, Android programming |

Server-side:

* Mac: Setting up and managing the server on a Mac can be done with flexible and versatile command-line tools. It offers decent control and customization options.
* Linux: Linux, like Mac, provides robust command-line tools for server setup and management. It's often a cost-effective choice, especially for hosting services.
* Windows: Windows offers a variety of software options for server management, making it user-friendly for those familiar with the Windows environment. However, it might have slightly more software choices compared to other OS options.
* Mobile Devices: When dealing with mobile devices, it's preferable to have a static server location that can be accessed from a single point. The server requirements for mobile devices are generally well-defined.

Client-side:

* Mac: The client-side setup on a Mac requires moderate expertise and time. The cost is comparable to Windows.
* Linux: Setting up the client-side on Linux demands the highest level of knowledge and skills but comes with the advantage of lower costs.
* Windows: Windows client-side setup is relatively straightforward, requiring minimal knowledge and time investment. Costs are like Mac.
* Mobile Devices: Mobile devices offer flexibility as potential clients and developers can access developments from anywhere. However, implementing client-side solutions on mobile devices may be slightly more challenging than on other platforms.

Development Tools:

Commonly used languages include JavaScript, CSS, and HTML, with supporting libraries for frontend development. General-purpose developer tools and languages such as Java, Ruby, Python, and PHP is also available.

* Tools: There are various development tools available, including PyCharm, Eclipse, Visual Studio, and GitHub. Additionally, text editors like Notepad++ and databases such as MongoDB, SQL, and Cassandra are essential for development.
* These languages and tools are accessible on all major operating systems, ensuring developers can work with their preferred environment regardless of the OS they use.

In simpler terms, the choice of operating system for server-side and client-side development depends on factors like expertise, cost, and platform-specific requirements. Development tools and languages are generally accessible across all major operating systems, providing flexibility to developers.

## 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**: After considering various aspects, it appears that Linux OS emerges as the most suitable operating platform for our project. This choice is primarily influenced by Linux's widespread acceptance as a server OS, coupled with its open-source nature. The open-source attribute not only makes it cost-effective at the initiation stage but also complements other potential solutions seamlessly. In conjunction, we can entrust the server management to a cloud provider, which would be a more economical option. Also, Linux demonstrates excellent compatibility with deploying the application across diverse platforms, enhancing its appeal as the preferred choice.
2. **Operating Systems Architectures**: Central to the Linux architecture is the Linux Kernel, which houses the fundamental operations of the operating system. Complementing this Kernel are various layers of the system, including the shell, which acts as the intermediary between the user and the Kernel. This arrangement grants us a substantial degree of adaptability, enabling us to tailor the shell optimally to cater to the specific requirements of our application.
3. **Storage Management**: For optimal storage management, utilizing a cloud computing service appears to be the most prudent approach, with the Google Cloud Storage Multi-Regional package standing out as the foremost option. This package not only assures remarkable performance for a vast and diverse client base but does so at a cost-effective price. Furthermore, it supports a variety of Linux OS alternatives, thereby accommodating the application's requirements seamlessly. Leveraging this external storage solution also paves the way for cost-efficient scalability, enhancing the project's flexibility and potential for growth.
4. **Memory Management**: Linux OS stands out as a proficient system in managing memory effectively, owing to its utilization of virtual memory. This means it only commits data to memory as necessitated by the process, enhancing the scalability of memory management. Consequently, this methodology is anticipated to foster heightened performance throughout the system, optimizing resource usage and facilitating smoother operations.
5. **Distributed Systems and Networks**: Leveraging a distributed system, characterized by its operation across various devices, is instrumental in this scenario, particularly considering the use of a cloud-based server cluster. This arrangement ensures that a multitude of devices can process requests from diverse platforms efficiently, facilitating a seamless and quality experience for all users involved. The extensive network infrastructure lays a solid foundation for delivering a premium product to the entire user base.
6. **Security**: In earlier versions, Linux OS encountered several security issues, but substantial improvements have been made over time to address these challenges. The cornerstone of its current security infrastructure is an authentication mechanism that requires users to provide verified credentials, such as a username and password, to access approved system segments. Contrary to the past, this information is not stored in publicly accessible files anymore, enhancing the overall security framework. Additionally, the integration of Pluggable Authentication Modules (PAM) has fortified the security structure further, offering a suite of adaptable security libraries that can be updated as necessary, ensuring a robust and up-to-date security environment.

All in all, we recommend that The Gaming Room adopts a cloud-based operating platform, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud, to facilitate the expansion of Draw It or Lose It across various computing environments. These cloud platforms offer scalability, flexibility, and streamlined deployment, making them ideal for accommodating diverse platforms. The choice of operating system architecture, whether virtual machines, containers, or serverless computing, should be tailored to the game's specific resource and scalability requirements. For storage management, we propose utilizing the cloud provider's native storage services, such as Amazon S3 or Azure Blob Storage, to ensure data durability and seamless integration. Memory management on cloud platforms is abstracted, allowing dynamic resource allocation. To enable communication between platforms, we recommend a microservices architecture with RESTful APIs or message queues while considering redundancy and fault tolerance. Security is paramount, and cloud providers offer robust security features, including IAM, encryption, and DDoS protection. Implementing user data protection measures, including authentication and encryption, is essential. Regular security audits and penetration testing should be conducted for comprehensive data security. This approach will enable a secure and scalable expansion of Draw It or Lose It across diverse computing environments.