

Draw It Or Lose It

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 09/15/2023 | Colin Aheron | Updated the executive summary, design constraints and domain model sections. |
| 2.0 | 09/30/2023 | Colin Aheron | Completed the evaluation section. |
| 3.0 | 10/15/2023 | Colin Aheron | Completed the recommendations section. |

**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**

The software design challenge at hand revolves around developing a web-based version of The Gaming Room's popular game, "Draw It or Lose It." The key requirements include enabling multiplayer gameplay with teams and players, ensuring unique game and team names, and maintaining a single active game instance in memory. To address this, we propose a solution that leverages the Singleton pattern to guarantee a single game instance, structured entity classes with unique identifiers for game and team names, a name uniqueness check, and comprehensive game round management.

**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**

One significant constraint is latency and network communication. In a distributed setup, network latency is inevitable, leading to delays in data transmission. To counteract this, the application must effectively handle asynchronous communication.

Scalability is another crucial constraint. The game's popularity may lead to varying levels of demand, necessitating the ability to accommodate a large number of concurrent users. Scalability considerations encompass architecture design, cloud-based resources, and load balancing techniques.

Security is a paramount concern in web-based distributed gaming applications. They are vulnerable to various security threats, including data breaches, DDoS attacks, and cheating. Robust security measures like encryption, authentication, and authorization are essential. Implementing anticheat mechanisms is critical to ensure fair gameplay.

Cross-browser compatibility poses a unique challenge, as players may access the game from various web browsers and devices, each with differing capabilities and standards compliance. Extensive testing and compatibility checks are required to ensure the game functions correctly across multiple browsers and devices.

Web browsers have resource constraints, such as limited memory and processing power. Developers must optimize the game's front-end code and assets to ensure smooth performance on various devices.

Lastly, ensuring a responsive and engaging user experience is crucial. Users expect a smooth gaming experience regardless of their device or location

In navigating these design constraints, a comprehensive and thoughtful approach to software development is essential. Prioritizing scalability, security, and user experience while addressing challenges related to latency, network communication, and compatibility is key to delivering a successful and enjoyable web-based game application.

**System Architecture View**

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**

ProgramDriver Class:

This class appears to represent the program's entry point with a main() method. It is not directly related to the other classes in the diagram but uses the SingletonTester class for testing purposes, as shown by the <<uses>> association.

SingletonTester Class:

This class is responsible for testing the Singleton pattern and contains a testSingleton() method. It is used by the ProgramDriver class to verify the singleton behavior of another class, the GameService class.

Entity Class:

The Entity class represents a general entity with attributes "id" and "name". It includes a constructor to initialize these attributes, getter methods for retrieving them, and a toString() method for generating a string representation.

GameService Class:

The GameService class manages the game-related operations and follows the Singleton design pattern, ensuring only one instance exists. This demonstrates the object oriented programming principle of encapsulation. It has attributes for managing game data (games) and unique identifiers (nextGameId, nextPlayerId, and nextTeamId). The class provides methods for adding games, getting games by ID or name, getting the count of active games, and managing player and team identifiers. The GameService class is connected to the Game class, representing that it manages games and has a list of active games.

Game Class:

The Game class represents a specific game instance and has a list of teams associated with it.

It includes a constructor to initialize the game with an ID and name and a method to add teams to the game. The toString() method generates a string representation of the game. The Game class is connected to the Team class, indicating that a game consists of one or more teams.

Team Class:

The Team class represents a team within a game and has a list of players.

It includes a constructor to initialize the team with an ID and name and a method to add players to the team. The toString() method generates a string representation of the team. The Team class is connected to the Player class, signifying that a team comprises multiple players.

Player Class:

The Player class represents a player within a team and includes attributes for ID and name.

It has a constructor to initialize player data and a toString() method for generating a string representation.

Object-Oriented Programming Principles:

- Encapsulation is used as classes encapsulate their data and provide controlled access through getter methods, adhering to the principle of data hiding.

- Inheritance is implied as the Game, Team, and Player classes seem to inherit common attributes or behaviors from the Entity class (e.g., id and name attributes).

- Abstraction is shown because the Entity class serves as an abstract representation of an entity with common attributes (id and name) and methods (getId(), getName(), and toString()). It encapsulates the common characteristics and behaviors shared by game-related entities (e.g., Game, Team, and Player).

- The toString() method, which is present in several classes in the diagram, exhibits a form of polymorphism. Polymorphism in this context refers to the ability of different classes (Game, Team, and Player) to provide their own implementation of the toString() method, allowing each class to represent itself as a string in it's own way.



**Evaluation**

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 is available but has been simplified in recent versions. Mac is not as commonly used for server deployments as Linux.  Advantages: Familiar development environment for Mac developers. Suitable for smaller-scale deployments.  Weaknesses: Limited server-specific features compared to Linux. May not be the best choice for scaling to thousands of players. | Linux is widely used for server-based deployments. It offers a range of server distributions (e.g., Ubuntu Server, CentOS) and web server software (e.g., Apache, Nginx).  Advantages: Robust, stable, and highly customizable. Excellent for hosting web applications. Strong support for web server software.  Weaknesses: Requires expertise to configure and maintain, especially for complex setups. | Windows Server is designed for server-based deployments and supports hosting web applications.  Advantages: Familiar interface for Windows administrators. Good integration with Microsoft technologies.  Weaknesses: Licensing costs can be substantial, especially for larger deployments. May not be as cost-effective as Linux. | Mobile devices, due to their limited resources and reliance on battery power, are not well-suited for hosting web-based software applications at scale. They can serve as portable testing environments for small audiences and provide emergency hosting in niche situations. However, their scalability limitations, potential reliability issues, and security concerns make them unsuitable for production-level hosting of high-traffic web applications. |
| **Client Side** | Supporting multiple types of clients on Mac entails additional development considerations. The cost may increase due to the need for Mac-specific development expertise and the potential for acquiring a range of Mac devices for testing and compatibility assurance. Development time should also account for adapting the software to different Mac models and macOS versions, potentially prolonging the development cycle. | Supporting multiple types of clients on Linux requires expertise in cross-distribution compatibility, which can increase development time and potentially lead to higher costs due to the need for testing on different Linux distributions and desktop environments. Ensuring that the software works seamlessly across various flavors of Linux demands thorough testing and adaptation, contributing to the project's overall time and resource requirements. Additionally, expertise in packaging and distribution methods for different Linux distributions is essential to effectively deliver and maintain the software, adding to development complexity and expertise demands. | Supporting multiple types of clients on Windows can involve significant software development considerations. Developing for different Windows versions and editions may require extra development time and expertise, potentially increasing costs. Additionally, ensuring compatibility with various hardware configurations, drivers, and user environments on Windows can be time-consuming, but it's essential for a smooth user experience, especially considering the diversity of the Windows user base. | Development costs may increase due to the need for platform-specific development (iOS and Android) and potential device fragmentation. It can also take additional time to develop and test for different screen sizes, resolutions, and operating system versions, requiring expertise in mobile app development. Overall, accommodating multiple mobile platforms requires careful planning, increased development resources, and expertise in both iOS and Android to ensure a consistent and optimized user experience. |
| **Development Tools** | Building software for deployment on Mac typically involves using programming languages like Swift and Objective-C, which are supported by Apple's Xcode integrated development environment (IDE). Developers often use Xcode to design, code, test, and package applications for macOS, utilizing additional tools like Interface Builder for user interface design and Instruments for performance analysis and debugging. | Building software for deployment on Linux typically involves programming languages like C, C++, Python, and Java. Commonly used development tools and integrated development environments (IDEs) include GCC (GNU Compiler Collection), Clang, Python's IDEs like PyCharm or Visual Studio Code with Python extensions, and Java development tools like Eclipse or IntelliJ IDEA, along with Linux-specific libraries and utilities for packaging and distribution. | Building software for deployment on Windows typically involves using programming languages like C++, C#, or .NET, and popular Integrated Development Environments (IDEs) such as Visual Studio. Other essential tools include version control systems like Git, build and deployment tools like MSBuild, and debugging tools to ensure compatibility and optimal performance on the Windows platform. | Building software for deployment on mobile devices typically involves using platform-specific programming languages and tools. For iOS, developers commonly use Swift or Objective-C with Xcode as the integrated development environment (IDE), while for Android, Java or Kotlin with Android Studio as the IDE are the prevalent choices. |

**Recommendations**

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

**Operating Platform**: Given the benefits of cloud-based server platforms, I recommend that The Gaming Room consider using a leading cloud provider such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). These providers offer a range of services that can accommodate the server-side needs of "Draw It or Lose It" and provide the scalability and cross-platform support required for expansion.

Additionally, the use of cloud platforms allows for seamless integration with various front-end technologies and platforms, including web browsers, mobile devices, and even integration with gaming consoles or PC ports. It provides a robust and future-proof solution for the game's server architecture.

* **Operating Systems Architectures**: Utilizing cloud services as the operating platform architecture for "Draw It or Lose It" involves hosting the game's core logic, assets, and computational processes on cloud infrastructure provided by service providers like AWS, Azure, or GCP. These cloud servers are distributed across data centers globally, ensuring low-latency access for players. Scalability is a key advantage, allowing you to dynamically adjust server resources in response to player demand, ensuring optimal performance. Communication between client devices and cloud servers is optimized for low-latency game streaming, making it suitable for a wide range of player devices. Cloud platforms also offer security measures, compliance options, and resource management tools, making them a flexible and robust choice for a diverse and potentially large player base.
* **Storage Management**: For the recommended cloud-based operating platform, an appropriate storage management system would be object storage. Object storage, offered by cloud service providers like Amazon S3, Azure Blob Storage, or Google Cloud Storage, is highly scalable and well-suited for storing game assets, media files, and player data. It offers low-latency access to data, which is essential for providing a seamless gaming experience. Object storage systems can be easily integrated with your game's server-side logic, providing efficient and cost-effective storage solutions for "Draw It or Lose It."
* **Memory Management**: Memory management techniques on the recommended cloud-based operating platform for "Draw It or Lose It" include dynamic resource allocation, resource pooling, caching, memory profiling, and garbage collection. These techniques optimize memory usage, ensure efficient allocation, and monitor memory in real-time.
* **Distributed Systems and Networks**: To enable cross-platform communication in "Draw It or Lose It," a distributed software architecture and robust network infrastructure are essential. The game client should be designed to run on multiple devices and operating systems to ensure cross-platform compatibility. The network must provide reliable connectivity and high-speed internet to minimize disruptions.

The server architecture should handle concurrent connections efficiently, utilizing load balancing, scaling, and failover mechanisms. Data synchronization, latency management, and security measures are critical to ensure smooth gameplay and data protection. Extensive cross-platform testing is necessary to identify and address any platform-specific issues, and centralized player data storage should maintain data consistency across platforms. In summary, a well-structured architecture and network setup are vital to facilitate communication between various platforms while managing dependencies and potential network outages.

* **Security**:To start, implement end-to-end encryption to safeguard user data during transmission. Use industry-standard encryption protocols like SSL/TLS to protect data exchanged between clients and servers, ensuring that it remains confidential and secure. Enforce robust user authentication mechanisms, such as multi-factor authentication (MFA) or OAuth, to verify the identity of users on different platforms. Employ role-based access control to limit data access to authorized personnel only. Utilize secure and encrypted storage solutions offered by the recommended cloud operating platform. These platforms often have built-in security measures to protect data at rest, including encryption, access controls, and compliance certifications. When displaying sensitive user data, implement data masking and anonymization techniques to replace or hide sensitive information. This ensures that only authorized individuals can access sensitive details. Keep software and systems up-to-date with the latest security patches and updates. Regularly review and apply security patches for your cloud infrastructure and server software. Finally, leverage the security capabilities of the recommended operating platform, including built-in security services and compliance features. Cloud platforms offer robust security measures, including access controls, firewalls, and encryption, which can be instrumental in securing user data.