Draw It or Lose It
CS 230 Project Software Design Overview
Version 4.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	01/21/2024	Alan Chumsawang	Initial Design
2.0	02/10/2024	Alan Chumsawang	Updated Evaluations
3.0	2/17/2024	Alan Chumsawang	Updated Evaluations
4.0	2/21/2024	Alan Chumsawang	Final Revisions

Executive Summary

Creative Technology Solutions goal is to develop their web-based version of The Gaming Room's present game, Draw It or Lose It. This project end goal is to bring the experience of Draw It or Lose It to cross platforms by utilizing a web-based environment. Our proposed solution involves multiple class identifiers for games, teams, and players, ensuring seamless gameplay and memory management. The use of design patterns, such as the singleton and iterator patterns, will enhance the application's user experience. Our development approach prioritizes unique team and player names, allowing users to check name availability when creating teams.

Requirements

1. **Cross-Platform Compatibility:**
    - The application must be accessible on multiple platforms.
2. **Team-Based Gameplay:**
    - The application must support multiple teams.
3. **Name Uniqueness Check:**
    - Team and player names must be unique, with a method to check name uniqueness during the creation of new teams, players, and games.

Design Constraints

1. **Web-Based Environment:** Application must be developed in a web-based environment to support cross-platform accessibility.

2. **Unique Identifiers:** Unique identifiers for games, teams, and players are critical to prevent conflicts, enable the one-instance-in-memory constraint and name uniqueness check mechanism.

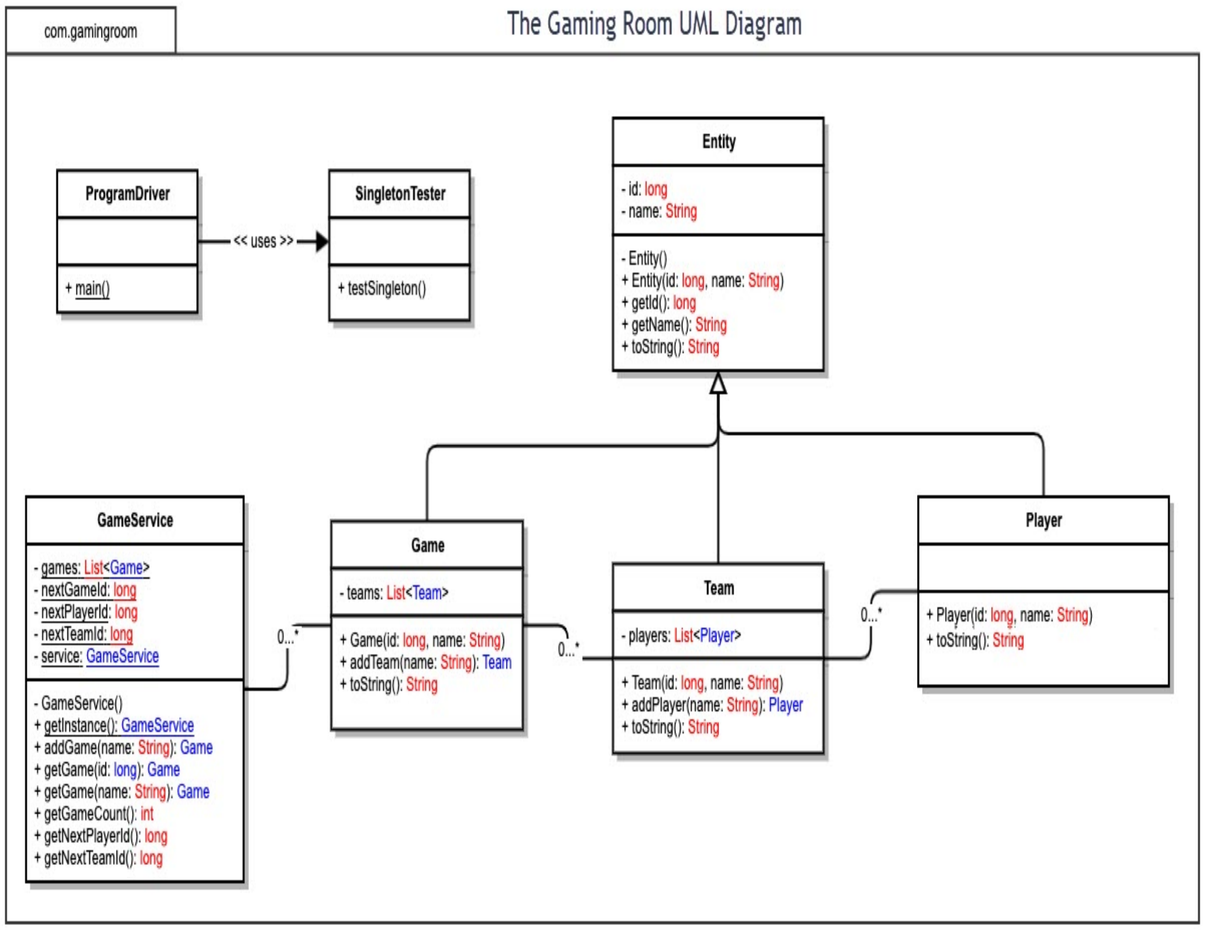
3. **Memory Management:** The application should manage memory efficiently, considering the one-instance constraint and the potential for a large user base.

4. **Design Patterns:** Incorporating the singleton pattern for the GameService class and the iterator pattern for adding and retrieving games enhances code structure and functionality.

System Architecture View

Domain Model

The UML class diagram provides a visualization of the relationships and structure of the game application's entities. The Entity class is the foundation for our attributes and behaviors, adhering to object-oriented programming principles. The Game class inherits from the Entity class, promoting code reuse and maintainability. The relationships between Game, Team, and Player classes are well-defined, demonstrating a clear representation of the game's domain model.



Evaluation

Development Requirements	**Mac**	**Linux**	**Windows**	**Mobile Devices**
Server Side	Mac is stability and secure, can be ideal for hosting with its macOS Server tools. However, scalability may be limited for larger applications. May have to consider third party apps to scale.	Linux is open source and is a preferred choice for server hosting. It excels in security, scalability, and supports a vast range of server applications.	Windows Server provides a user-friendly environment for hosting web applications. It supports various web servers and tools for application deployment.	Mobile devices are not primary servers, lacking the hardware for large applications. However, they can play a role in client interactions.
	- Mac offers server-based deployment methods.	- Linux is ideal for server-based deployment with a wide range of hosting options.	- Windows provides server-based deployment	- Mobile devices are not suitable for hosting but can interact with the server.
	-Mac has licensing cost.	-Linux is free and open source.	-Windows has licensing costs.	-No server licensing costs with limited access
Client Side	Considerations of development costs, time, and expertise. Mac users expect a seamless and visually appealing experience. Using MacOS native apps may be necessary to achieve this.	Diverse Linux clients demands compatibility considerations. Lower development cost. However, must find Linux expertise.	Highly used through the common market which means there are a considerable number of established tools.	Mobile devices demand considerations for different OS results in potential high development costs.
Development Tools	For deploying on Mac, developers commonly use XCode as the IDE, supporting multiple languages.	Linux development involves diverse tools like Visual Studio Code, Eclipse supporting a vast number of languages.	Windows development commonly utilizes Visual Studio as the primary IDE, supporting many languages.	Development for mobile devices involves platform-specific IDEs like XCode for iOS and Android Studio for Android.
	Knowledge of XCode required. Licensing costs may be required.	Open source, free, most diverse tool library.	Visual studio knowledge required; licensing cost may be required.	Teams may have to be separated by device operating system.

Recommendations

1. Operating Platform:

   Recommendation: Utilize a web based operating platform for cross-platform accessibility.

   Justification: Web based platform provides potential of cross-device compatibility, meeting the project's cross-platform requirements. This also aligns with the design constraint of developing the application in a web-based environment.

2. Storage Management:

   Recommendation: Utilize cloud-based storage, such as AWS or Azure, to allow horizontal and vertical scaling.

   Justification: Cloud based storage provides required scalability and reliability for a web-based application with high traffic. This aligns with the design constraint of efficient memory management and supports the goals of scalability and low-latency storage.

3. Memory Management:

   Recommendation: Utilize memory management capabilities of the web-based operating platform.

   Justification: Effective memory management is crucial for a web-based game, especially considering potential large user bases. Leveraging the capabilities of the chosen operating platform and implementing efficient algorithms will enhance the overall performance and user experience.

4. Distributed Systems and Networks:

   Recommendation: Utilize microservices architecture for Draw It or Lose It. Use APIs for communication between different components, allowing seamless integration and scalability.

   Justification: Microservices architecture enables seamless scalability and ease of maintenance. Utilizing APIs for communication ensures efficient communications, aligning with the design pattern recommendations and design constraints.

5. Security:

   Recommendation: Utilize web-based operating platforms with robust security features, regular updates, and strong user authentication system to protect user information across platforms.

   Justification: Security is a critical concern for web-based application. Encryption and a secure operating platform contribute to protecting user data, aligning with the design constraints and ensuring a secure gaming environment.