Draw It Or Lose It

CS 230 Project Software Design Document

Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.2	12/10/24	Owen McCostis	Updated ‘Recommendations’ section.
1.1	12/1/24	Owen McCostis	Updated ‘Evaluation’ section.
1.0	11/17/24	Owen McCostis	Created document and initialized sections.

Executive Summary

The Gaming Room aims to expand its Android game, *Draw It or Lose It*, by redeveloping it as a web-based application to increase accessibility to other platforms. I recommend the development of a scalable, distributed web application that accommodates one or many teams with multiple players on each team, ensures game and team names are unique, ensures there is only one instance of a game at a time, includes unique identifiers for each game, team, and player instance, includes a system to check for and enforce repeat instance names, and that is prepared to be integrated with future hardware requirements.

Requirements

Business Requirements:

- Develop a web-based version of *Draw It or Lose It*, which can be run on many platforms
- Allow one or more teams to participate in each game session
- Assign multiple players to each team
- Game and team names must be checked for uniqueness when their names are set to avoid duplicates
- Maintain only one instance of the game in memory at any given time

Technical Requirements:

- Implement unique identifiers for each game, team, and player instance
- Design a system to check and enforce the uniqueness of game and team names
- Create a distributed application architecture suited for web-based deployment
- Be aware of future hardware requirements throughout development

Design Constraints

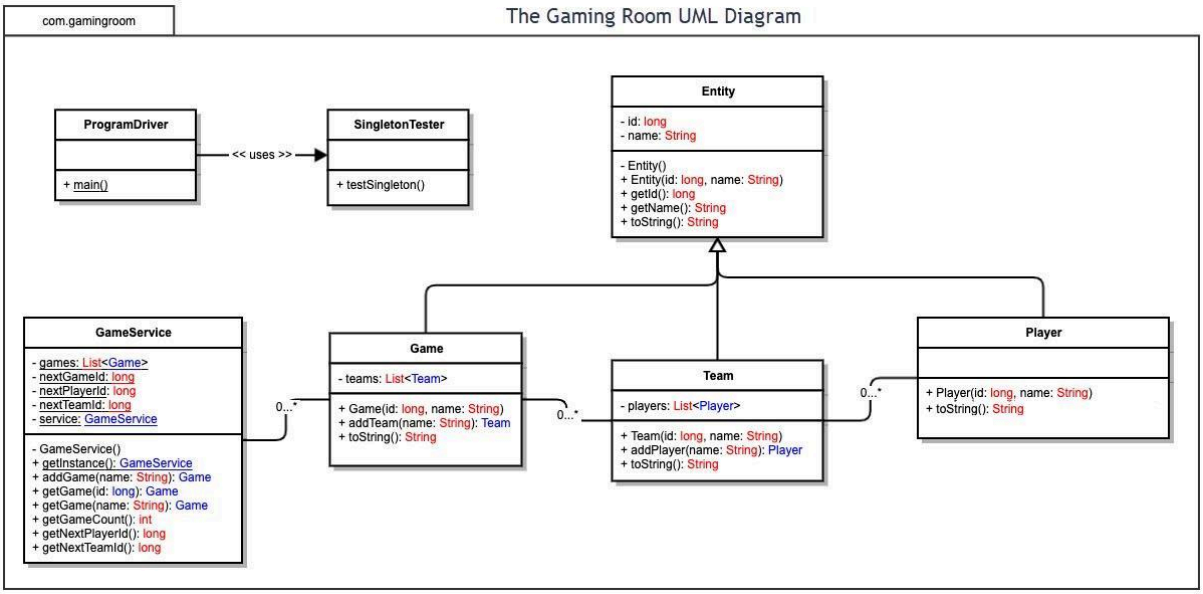
1. **Multiple Teams and Players**
    a. **Constraint:** The game must support one or multiple teams, each with multiple players
    b. **Implications:**
        i. The system must efficiently handle dynamic team creation and player assignment
        ii. Data structures must be optimized for quick access and updating
        iii. Concurrency control must be included to handle multiple players performing simultaneous actions
2. **Uniqueness of Instance Names**
    a. **Constraint:** Game and team names must be unique across the entire system
    b. **Implications:**
        i. Requires a centralized database or service to track and validate the uniqueness of names
        ii. Requires efficient querying mechanisms to check name availability without affecting performance
        iii. Increases the complexity of the naming logic to prevent collisions in a distributed environment
3. **Single Instance in Memory**
    a. **Constraint:** Only one instance of the game should exist in memory at any given time
    b. **Implications:**

i. Requires the implementation of a Singleton design pattern or similar control mechanism
ii. Requires careful session management to maintain consistency for all users
iii. Challenges arise in synchronizing states across multiple servers or instances in a distributed system

4. **Web-Based Distributed Environment**
    a. **Constraint:** The application must operate seamlessly over the web across various platforms
    b. **Implications:**
        i. Cross-platform compatibility requires the use of standard web technologies like HTML5, JavaScript, and CSS3.
        ii. Network reliability and varying client capabilities must be considered to ensure consistency
        iii. Requires security measures to protect data transmission and prevent unauthorized access in a distributed network
5. **Performance and Timing**
    a. **Constraint:** Game rounds are time-sensitive, with strict one-minute durations and precise image rendering
    b. **Implications:**
        i. Requires low latency to synchronize timers and events for all players
        ii. Requires efficient rendering techniques to display images progressively up to the 30-second mark
        iii. May require real-time communication protocols to handle rapid updates and user interactions
6. **Future Hardware Integration**
    a. **Constraint:** Hardware requirements will be determined in the future
    b. **Implications:**
        i. The software design must not be specific to any specific hardware to accommodate future changes
        ii. Requires modular architecture to allow integration with various hardware components when specified
        iii. Anticipating potential hardware capabilities and limitations will enable development flexibility

Domain Model

- The Entity class serves as the parent class for Game, Team, and Player, encapsulating shared properties such as id and name. This promotes code reuse and reduces redundancy because child classes can inherit shared properties.
- The GameService class is designed to manage the system's state and ensure only one instance of the game exists at any time. The class is responsible for maintaining a list of games, generating unique identifiers for games, teams, and players, and ensuring unique names for games and teams.
- An instance of the Game class can have multiple instances of the Team class, which is a one-to-many relationship. The Game class is responsible for managing a collection of teams participating in the game.
- An instance of the Team class can have multiple instances of the Player class, which is also a one-to-many relationship. The Team class is responsible for maintaining a list of players and adding new players to the team.
- The Player Class represents individual players in the game and inherits id and name from the Entity class



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	**Characteristics:** MacOS servers are not typically used for large-scale web hosting as compatible server software is more limited. **Advantages:** A user friendly environment for developers accustomed to MacOS that can be easily integrated into an Apple ecosystem. **Weaknesses:** Less community support for server deployments. Mac hardware is more expensive.	**Characteristics:** Widely used for web server hosting and offers various distributions. **Advantages:** Stable and open-source. Often free or low cost, just requires modern browsers. **Weaknesses:** Requires more technical expertise to manage and configure.	**Characteristics:** Windows Server supports IIS for hosting web-based applications. **Advantages:** Familiar UI for administrators, strong support, and easy integration with Microsoft .NET and Microsoft Azure. **Weaknesses:** Costs can be higher and may require more resources to achieve comparable performance to other options.	**Characteristics:** Mobile devices do not serve as traditional web servers. Mobile server applications would be hosted elsewhere (like Linux or Windows) and accessed from mobile browsers or native apps. **Advantages:** Mobile devices are not applicable for hosting. **Weaknesses:** Mobile devices cannot be used to host web-based applications.
Client Side	Applications should be tested thoroughly on Safari and other MacOS browsers, ensuring support for Apple's ecosystem. Additional time may be needed to optimize performance. Developers must be familiar with Mac environments.	Applications should be responsive and standards-based, using HTML5, CSS3, and JavaScript. Minimal additional time required if following web standards. Developers must be familiar with Linux environments.	Applications should be tested on Windows browsers (like Edge, Chrome, and Firefox) and optimized for different screen resolutions and input methods. Additional time may be necessary to address compatibility issues. Developers must be familiar with Windows environments.	Applications should use responsive web design, CSS frameworks, and mobile testing tools. Various mobile devices should be tested continuously throughout development. Additional development and testing time for optimization. Developers must be familiar with mobile web development and native app development languages.

Development Tools	Swift or Objective-C are used for MacOS applications and HTML5, JavaScript, and CSS3 are used for web-based applications. Xcode is primarily used for native MacOS development.	Linux supports development in many languages like Java, Python, JavaScript, PHP, and Ruby. Many IDEs are also supported, like Eclipse, PyCharm, and Visual Studio Code.	C# and Visual Basic are primarily used for Windows-specific applications. Visual Studio is known for its robust development features. IIS is used for hosting web applications.	Java is primarily used for Android development, while Swift is primarily used for iOS development. JavaScript can be used for cross-platform development. Android Studio and Xcode are used for publishing mobile apps.

Recommendations

1. **Operating Platform**: I recommend deploying *Draw It or Lose It* on Linux because Linux servers are known for their scalability, robustness, and low cost. Using a web-based architecture, the game could be accessible from any platform compatible with modern web browsers, like Windows, Mac, Linux, iOS, and Android.

2. **Operating Systems Architectures**: Linux features a modular monolithic kernel architecture, which enables performance optimization and extensive configuration. Modules can be loaded/unloaded as needed, improving speed, security, and stability. Additionally, the modularity of the kernel architecture makes updates and maintenance simpler and easier.

3. **Storage Management**: An appropriate storage management system for this platform would be a relational database management system like MySQL, which is ideal for structured data and relationships between entities like the ones between users, teams, and games. A Content Delivery Network (or CDN) could be integrated for the image library to make image loading more efficient.

4. **Memory Management**: Linux employs many advanced memory management techniques. Virtual memory management isolates applications in memory and can effectively manage memory allocation during high-demand tasks. Paging and swapping is used to free up memory by temporarily moving 'pages' or 'blocks' of the system's RAM into storage. Demand paging reads only necessary data from storage, improving memory demand and load speeds.

5. **Distributed Systems and Networks**: The game operates on a client-server model where clients interact with the server hosting the game logic. Load balancing can distribute network traffic to multiple servers, ensuring demand is evenly spread out across available servers. This approach allows for the implementation of additional servers to handle increased load. Data transmission can be optimized to reduce delays. Handling partial failures and implementing retry mechanisms will maintain a seamless user experience. Redundant/backup servers can be deployed to mitigate the impact of outages. Backup systems can be automatically switched to in the event of a failure. System health should be continuously monitored to detect and address issues efficiently.

6. **Security**: TLS encryption can be used for all client-server communications to prevent interception. Sensitive data stored in the database should be encrypted using strong encryption algorithms. Unauthorized data access should be restricted by well-defined roles/permissions and robust authentication protocols. Injection attacks should be prevented by validating and sanitizing all user inputs. Network traffic should be continuously monitored for suspicious activities so attacks can be effectively identified and addressed. Ensuring software and operating systems are updated to the latest versions lessens the risk of attacks and data breaches.