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	07/21/24	Zackary Nelson	Initial creation and completion of the software design document

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 Gaming Room aims to develop a web-based version of their existing Android game, "Draw It or Lose It," which allows multiple platforms to access the game. The game involves teams competing to guess what is being drawn within a time limit. Our proposed solution includes creating a software design document and developing the game application to meet specified software requirements. This document outlines the constraints, architecture, and implementation plan to achieve a scalable, secure, and efficient game application.

Requirements

The client requires the game application to support multiple teams, each with multiple players. Team and game names must be unique, and only one instance of the game can exist in memory at any given time. The application should be web-based and support distributed environments to accommodate various platforms.

Design Constraints

Developing the game in a web-based distributed environment imposes several design constraints:

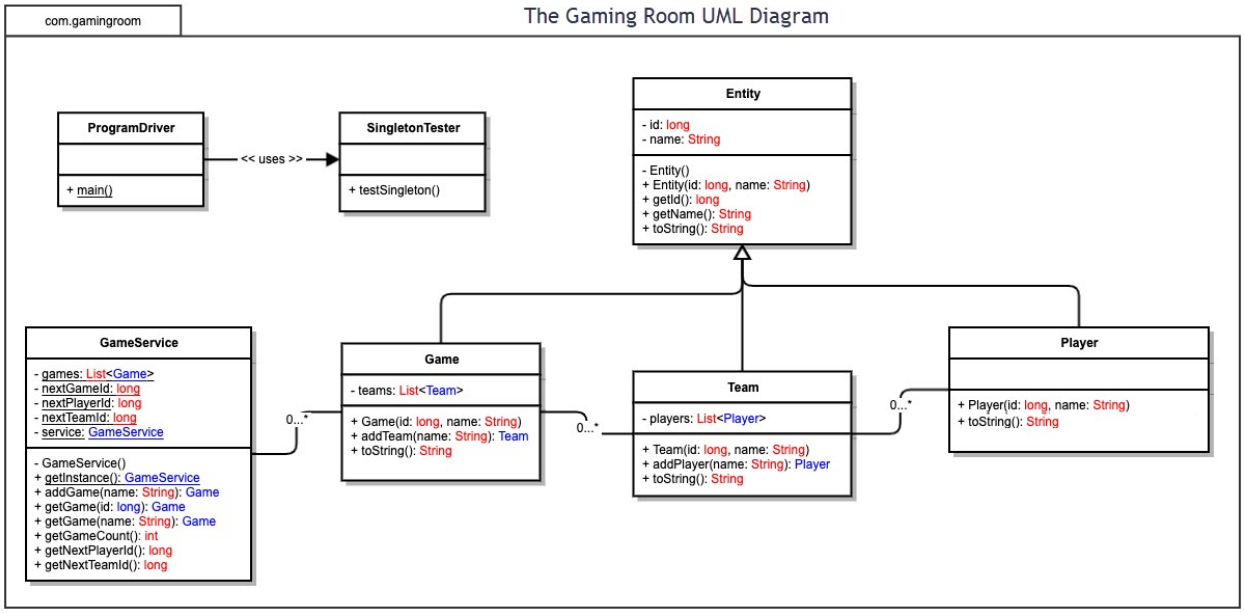
- **Scalability**: The application must handle multiple teams and players efficiently.
- **Unique Identifiers**: Unique names for games and teams to avoid conflicts.
- **Singleton Pattern**: Ensuring a single instance of the game exists in memory.
- **Platform Compatibility**: The application must be accessible across different operating systems and devices. These constraints will influence the development process, requiring careful planning and implementation of design patterns.

System Architecture View

No specific requirements for this section in this project. However, the system architecture typically includes a description of system and subsystem architecture, physical components, tiers, logical topology of communication, and storage aspects.

Domain Model

The UML class diagram represents the structure of the game application. The primary classes include `Game`, `Team`, `Player`, and `Entity`. The `Entity` class serves as a base class with common attributes (`id` and `name`). The `Game` class manages the game logic, `Team` class represents a team of players, and `Player` class represents individual players. Object-oriented principles such as inheritance (from `Entity` to `Game`, `Team`, and `Player`) and encapsulation are used to efficiently manage and access the game data.



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	Mac servers are robust but less common in enterprise environments. They provide a stable Unix-based environment.	Linux is highly regarded for server environments due to its stability, security, and open-source nature.	Windows servers are widely used and offer good support but are often seen as less stable than Linux.	Mobile devices typically rely on backend services hosted on servers rather than serving as servers themselves.
Client Side	Developing for Mac clients requires expertise in macOS-specific development tools and languages, which can be costly.	Linux client support may require specific expertise, and the user base is smaller compared to other platforms.	Windows has a large user base, and development tools are widely available, making it cost-effective.	Mobile development requires targeting specific OS (iOS, Android) and can be costly due to platform-specific requirements.
Development Tools	Mac development utilizes tools like Xcode and languages like Swift.	Linux development often uses tools like GCC, Clang, and IDEs like Eclipse.	Windows development uses tools like Visual Studio and languages like C#, C++.	Mobile development uses IDEs like Android Studio for Android and Xcode for iOS, with languages like Java, Kotlin, Swift.

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

I recommend using Linux for server-side deployment due to its stability, security, and cost-effectiveness. Linux is widely used in server environments because it's open-source, has strong community support, and provides flexibility in customization. For client-side, support should be provided for Windows and mobile devices (iOS and Android) because they have large user bases, ensuring the game reaches a broad audience.

2. Operating Systems Architectures

The chosen operating platform architectures should include Linux for the server environment and support for Windows, iOS, and Android on the client side. This approach ensures broad accessibility and leverages the strengths of each platform. For example, Windows is known for its user-friendly interface, while iOS and Android are optimized for mobile performance. Each platform requires careful consideration of how system resources are managed, particularly in handling different user interface designs and resource constraints.

3. Storage Management

For storage management, I recommend using a cloud-based solution like Amazon S3 or Google Cloud Storage. These options provide scalable and reliable storage that can handle large amounts of data. Additionally, they offer redundancy and backup options to ensure data is always available and secure. Considering the game's data, these cloud services are equipped to manage issues like latency and data consistency effectively.

4. Memory Management

Linux's memory management techniques, such as swapping and efficient allocation, will ensure that the game application runs smoothly on the server. On the client side, it's important to use efficient data structures and algorithms to optimize memory usage, especially on mobile devices where resources are limited. For example, minimizing the memory footprint and using techniques like garbage collection in Java (for Android) will help the game perform better across different platforms.

5. Distributed Systems and Networks

To enable communication between different platforms, I suggest implementing RESTful APIs, which are widely used for their simplicity and scalability. Additionally, adopting a microservices architecture can help manage dependencies between various components, allowing different parts of the system to scale independently. This approach also helps in handling potential issues like network latency and service interruptions by isolating services and reducing the impact of failures.

6. Security

Security is crucial, especially when dealing with user information. I recommend implementing HTTPS to secure communication between clients and servers, ensuring data is encrypted during transmission.

Additionally, sensitive data should be encrypted at rest, using strong encryption algorithms. Regular security audits are essential to identify and fix vulnerabilities. It's also important to implement secure user authentication methods, such as OAuth2 or multi-factor authentication, to protect user accounts. If the game will be available internationally, compliance with data protection regulations like GDPR (in Europe) or CCPA (in California) should also be considered.