THE GAMING ROOM

Draw It or Lose It

Software Design Template

Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/21/25	Kimani Muhammad	Draw It or Lose it Design Template initial creation.
2.0	08/03/25	Kimani Muhammad	Completed the OS evaluation.
3.0	08/14/25	Kimani Muhammad	Added the final recommendations.

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

Creative Solutions Technology is proposing to create a game named Draw It or Lose It. The game has four rounds, giving a team 30 seconds to solve the drawing. If the time expires, the remaining teams have a chance to guess with a 15-second time limit. This is where Creative Solutions Technology is having trouble setting up the game environment. I'm proposing that we use the Singleton model to ensure that one instance of the game exists that is assigned to the next team, allowing multiple teams to share a single game.

Requirements

- A game will have the ability to have one or more teams involved.
- Each team will have multiple players assigned to it.
- Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
- Only one instance of the game can exist in memory at any given time.

Design Constraints

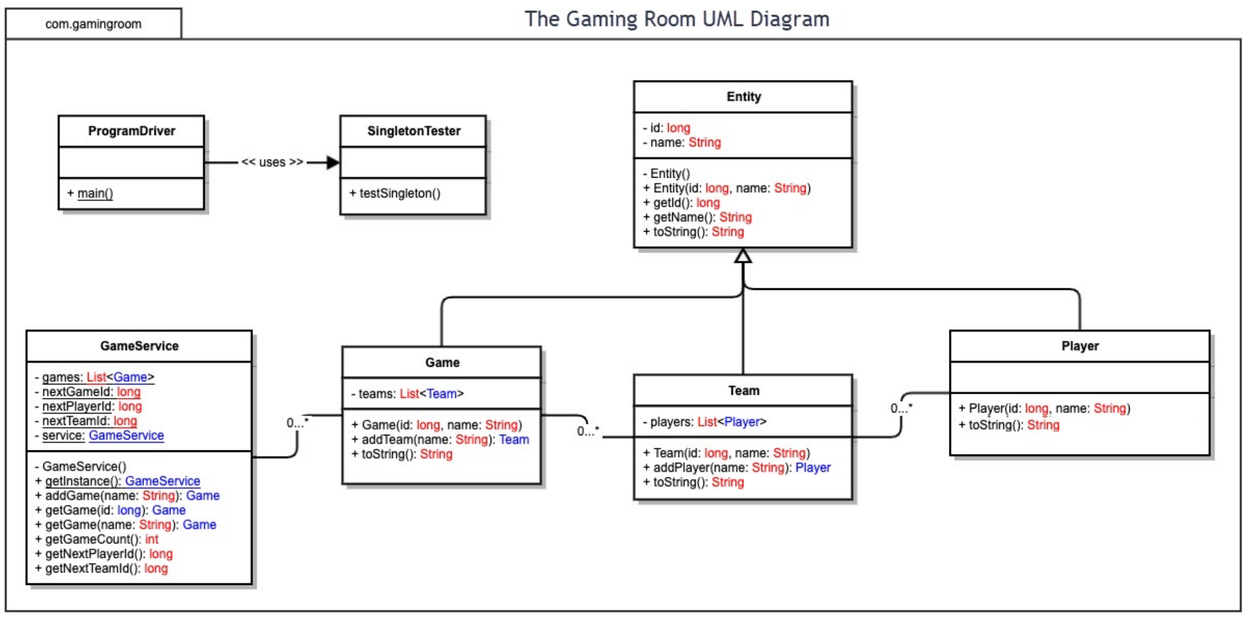
The game will need to be implemented using a Singleton design pattern, so only one instance of the game will be able to exist at any given time. The code will be best developed in Java using Object-Oriented Programming. Each game will require a unique name with the ability to have one or more teams with unique team names. Each team will have multiple players.

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

The program begins with the main function in the ProgramDriver class. The ProgramDriver uses the SingletonTester class to test the Singleton design pattern using the testSingleton() method. The Entity class is implemented as a parent class, demonstrating encapsulation by holding the common private fields and public fields and methods. The Entity class is inherited by the Player, Team, and Game classes, adding their own class-specific private and public fields and methods. This also demonstrates polymorphism as each of the three mentioned classes can be described as an Entity. The GameService class can only have one instance and has a 0 to many relationship with the Game class, giving the game service the ability to have multiple games. Abstraction is demonstrated in the GameService class as only the necessary methods are made available to the user while the fields are kept private. The Game class also has a 0 to many relationship with the Team class, giving the game the ability to have multiple teams. Finally, the Team class has a 0 to many relationship with the Player class, giving each Team the ability to have multiple players.



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 has a Unix-based operating system that's best for development environments but lacks scalability in production. It has a user-friendly GUI, but macOS devices are not designed for continuous uptime, and cloud providers don't provide MacOS instances for web app hosting.	Linux is open source so it's free. It also supports most modern tech stacks, scalable, and efficient due to modularity. It has strong security with quick vulnerability patching. However, it's more complex to use due to minimal GUI and limited support.	Windows has excellent performance hosting Windows-native apps, but less efficient for hosting other applications. It has great active directory integration and patching system, but it also requires regular security maintenance. The licensing is paid with higher infrastructure costs making it a more costly option. Its GUI is beginner-friendly, making it easy to use.	Android lacks a firewall and has poor SSH support. Android also has no DNS resolution with ports being blocked by mobile carriers. iOS doesn't allow apps to listen to incoming network traffic and no background service processed are allowed. Hosting on mobile devices is mostly only theoretical.
Client Side	To support multiple clients on Mac, Mac hardware is needed for testing. If not developing on a Mac, the testing and development will also be much slower. It also requires a high level of expertise since Safari has different behavior than expected with CSS, JS, and WebGL.	Linux is low cost due to ability to run on virtual machines or cheap hardware. Linux also requires minor knowledge of Linux desktop environments and comfort with Linux browser bugs. Being able to run on a VM with minimal knowledge of Linux means faster development time and less cost overall.	Windows requires hardware but is also cheaper and more common. Chrome, Firefox, and Edge can be used for testing for free on local machines. Windows also has faster development and time to support due to having standards compliant web browsers and less extra development. Also, minimal expertise needed for development as Windows is more common.	For iOS, Mac hardware is required to test iOS Safari. Another con is that only some interactions can be tested. iOS has strict limitations and requires custom fallback behavior for unsupported features. iOS also required deep expertise of iOS Safari limitations and constraints. Android also has a high cost due to needed multiple real devices or emulators for testing. Android has slower development since you have to test across a variety of screen resolutions and input types. Android also requires advances expertise for responsive web design and performance profiling.

Development Tools	IDEs used for Mac include VS Code, WebStorm, Sublime Text, and Xcode. Mac uses Safari, Chrome, and Firefox browsers for testing. Relevant programming languages include JavaScript, TypeScript, and HTML5. Other relevant tools include Charles Proxy and Safari DevTools.	IDEs used for Linux include VS Code, Atom, GNOME Builder, and Vim/ Emacs. Linux uses Chrome and Firefox browsers for testing. Relevant programming languages include JavaScript, TypeScript, and HTML5. Other relevant tools include Gnome Terminal, DevTools, and XDebug.	IDEs used for Windows include VS Code, WebStorm, and Sublime Text. Windows uses Chrome, Edge, and Firefox browsers for testing. Relevant programming languages include JavaScript, TypeScript, and HTML5. Other relevant tools include Fiddler, DevTools, and Windows Terminal.	IDEs used for iOS include Safari and Chrome. iOS uses Xcode Simulator and Safari's Develop Tools for testing. Relevant programming languages include JavaScript and HTML5. Both a Mac and iOS device are needed for full testing. IDEs used for Android include Chrome, Samsung Internet, and Firefox. Android uses Android Studio and Chrome Remote Debugging for testing. Relevant programming languages include JavaScript and HTML5. Other relevant tools include ADB and Genymotion.

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**: The recommended operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments is Linux for the server hosts. Linux is open-source, compatible with most tech stacks, has great security, and has the ability to run on virtual machines, making it the most affordable, efficient, and modular operating system to host the application and database server. This will allow the clients to remain cross-platform while uniformly connecting to the server.

2. **Operating Systems Architectures**: The Linux operating system is made up of the hardware, kernel, shell, and application layers. The hardware being the physical computer components such as the CPU and RAM. The core of the Linux operating system is the kernel. The kernel directly interacts with the hardware to manage resources such as memory, processes, and the file system. The shell is the interface that connects the user and the kernel by interpreting user

commands and turning them into kernel instructions. The most outer layer of the Linux operating system is the applications layer, which consists of user-level programs that call libraries which in turn invoke the kernel, such as the web browser.

3. **Storage Management**: Storing the images as JPEG images and using object storage such as AWS S3, and using a content delivery network to cache the files on edge servers for less latency and faster retrieval times. We'll also use a relational database such as Postgres to track the game state and data.

4. **Memory Management**: Since loading all 200 images into RAM would be inefficient, the application should use virtual memory in conjunction with demand paging. This approach loads only the images required for the current game round into memory, minimizing memory usage and reducing the likelihood of thrashing. We can accomplish this by using on-demand streaming with an LRU cache to hold the most recently accessed images. Since Draw It or Lose It is a heavily graphics-intensive application, efficient graphics processing unit (GPU) memory management is an important aspect of the application to consider to prevent slow rendering. Allocating GPU resources efficiently ensures images are displayed rapidly as required by the game application design. To accomplish this, we should use compressed textures, mipmaps, and texture atlases to minimize the GPU memory and release textures after use.

5. **Distributed Systems and Networks**: There will be stateless application servers running behind a load balancer to prevent outages by redirecting traffic to healthy backup servers if one is ever experiencing an outage to minimize any downtime. Using AWS, we can also maintain redundancy by distributing the instances across multiple availability zones. We should also use WebSockets for real-time gameplay events since it features a persistent connection, two-way communication, and low latency, and sends small data packets. These characteristics make WebSockets the perfect candidate since we need close to real-time response times when players are playing together. For non-real-time responses, we can use a REST API with HTTP methods.

6. **Security**: Follow the rule of least privilege across AWS IAM, servers, and databases. Enable and configure firewalls and blocking unused ports to limit unwanted outbound and inbound traffic to AWS EC2 instances, if cloud hosting. Only allow the hosts to communicate with the client and use TLS to prevent eavesdropping and perform penetration testing. Use JSON Web Tokens to authenticate user signing in to the game using securely stored, short-lived tokens and protect user data by hashing the passwords inside the database using Argon2id with password hashing. Apply OWASP best practices and DDoS protection. Use a secrets manager to manage the application secrets. Set up multi-factor authentication (MFA) for administrator and user access.