Draw It or Lose It

CS 230 Project Software Design Template

Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/5/22	Utkarsh Singh	Initial proposal
1.1	11/11/22	Utkarsh Singh	Added Evaluation
1.2	11/13/22	Utkarsh Singh	Added Requirements
1.4	12/9/22	Utkarsh Singh	Added Recommendations

Executive Summary

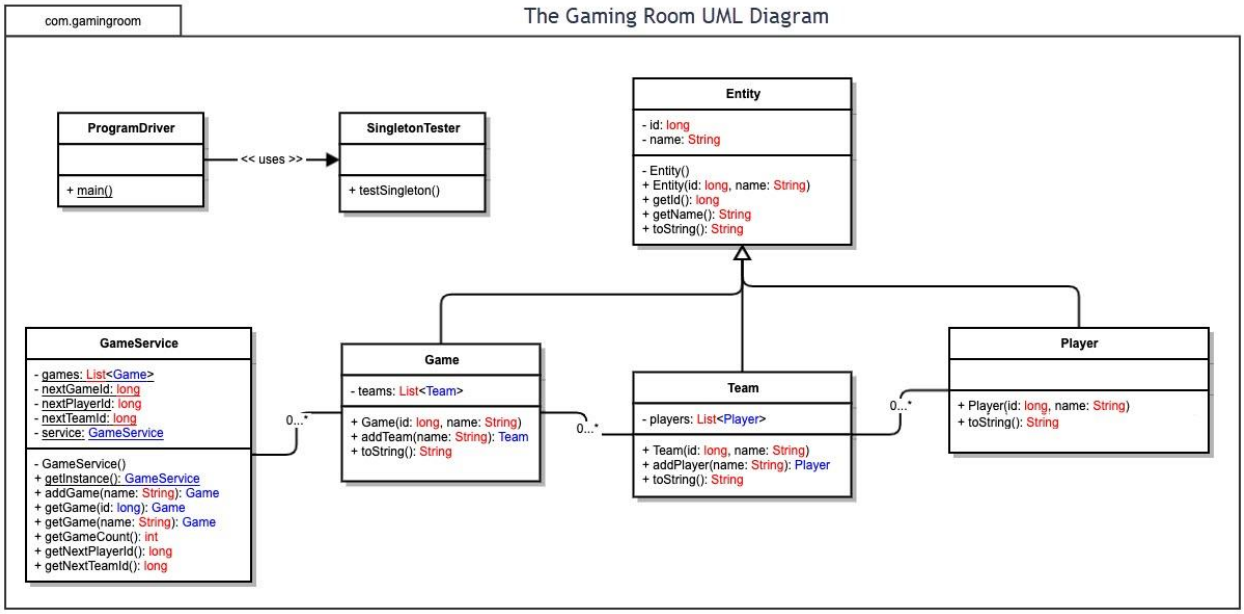
In a web browser, the game Draw It or Lose It aims to replicate the original gameplay. One team will try to guess the identity of an image that the application has successfully drawn. The drawing will be finished after thirty seconds, and the other teams will have fifteen seconds to identify the image.

Design Constraints

Since the game relies on real-time rendering, the application must have a pretty low latency between the servers and clients. A game will have the option of featuring one or more teams. There will be several players assigned to each side. To enable users to check whether a name is already in use when choosing a team name, game and team names must be distinct. At any given time, there can only be one instance of the game running in memory. To do this, distinct IDs can be made for each iteration of a game, team, or player.

Domain Model

The application's main method is contained in the ProgramDriver class, which also performs a singleton test using a SingletonTester instance of the GameService class. Multiple Game classes may exist for each GameService (if any). The GameService and Game classes have the following zero to many connection. Additionally, a Game may contain 0 or more Team classes and 0 or more participants may make up a Team. The Entity class serves as the common ancestor for the Game, Team, and Player classes. These classes feature additional variables and methods in addition to the built-in variables and methods of Entity, which are exclusive to its namesake.



Evaluation

Development Requirements	Mac/OSX	Linux	Windows	Mobile Devices
Server Side	Although running our web server on OSX would be substantially more expensive than on Linux or Windows, the idea would remain the same (configuring tomcat or another web server).	Depending on our deployment strategy, web server hosting will require the least amount of configuration and would be the most affordable.	Windows provides internal web server hosting through IIS, but the license is more expensive than Linux, which is available for free.	Hosting a web server on a mobile device is not logical. We could probably find out how to host a Kubernetes cluster on Android, but it would take some time, and iOS won't likely permit it.
Client Side	We must ensure that our online application is compatible with both Safari, the OS X native browser, and the other widely used browsers.	The two main browsers used in the Linux environment will be Firefox and Chrome.	Since Microsoft Edge is Windows' default browser, we must ensure compatibility with it as well as other widely used browsers like Google Chrome, Opera, Firefox, and so on.	The two leading competitors for users of mobile browsers are Chrome and Safari. Our efforts should be concentrated on mobile platform optimization.
Development Tools	If we want to deploy natively on OS X, we'll probably need to use XCode and Swift. If at all possible, we ought to avoid time-consuming tasks, use Java, deploy to Tomcat, and select an IDE (VSCode or Eclipse). There would be no fees for a license.	Tomcat should be deployed, along with VSCode and Java. In terms of cost and scalability, deployment directed at a Linux container would be the most efficient.	For our development pipeline, we can use Tomcat, Java, and Visual Studio Code;	If we wanted to host a webserver from a phone, we would need to utilize Android Studio and Kotlin/Java or XCode and Swift on another operating system (OSX, Unix, Windows). Even though there are no license fees, the expense would not be justified.

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**: **Operating Platform**: I recommend the Gaming Room to use Linux servers. I suggest creating a mobile app for iOS devices to expand the user field.

2. **Operating Systems Architectures**: Although Linux is a little bit more advanced its cost-friendly, reliable, and it is opensource so they can do more with Linux. It has great scalability.

3. **Storage Management**: Linux doesn't need a lot of storage. IaaS implementation is advised. "High-level APIs used to dereference many low-level characteristics of the underlying network architecture, like physical computer resources, location, data partitioning, scaling, security, backup, etc." are provided by IaaS. An example of a cloud computing service is IaaS. It is excellent for managing storage. One of the best cloud computing services is Microsoft Azure.

4. **Memory Management**: Powerful and quick is Linux. Excellent memory management. IaaS is excellent for managing memory. This is because it makes system management easier while providing storage, backup, and recovery.

5. **Distributed Systems and Networks**: To make sure that mobile device apps run on various operating systems, it's crucial to use a variety of programming languages. The program could connect to more customers using a server with good scalability and affordability, like Linux servers. Having multiple servers is a great way to ensure the website is always available as the number of users grows. Using a load balancer is a good idea to optimize response time and prevent overloading. RAID is a beautiful way to give memory and storage redundancy. For striping and parity, I suggest RAID 5. RAID 5 can tolerate a single drive failure. Or they might employ RAID 10. This RAID configuration is nested. It is also helpful because it uses striping to speed up data transfers and duplicates all the data on a second drive to keep it safe.

6. **Security**: When developing the front end of applications, it is critical to keep security in mind. Programmers must be sure to validate user input to prevent online threats like SQL injection and buffer overflow. It is wise to remember that C++ is vulnerable to buffer overflow attacks. A user with nefarious intentions can use SQL injections to access database information. They use the areas where the application inputs data to accomplish this. To keep people from taking advantage of your code, I recommend using the best coding practices on both the client and server sides. A root certificate must be obtained from a reputable certificate authority before the site may go live. This public key certificate demonstrates that the requested website is legitimate. This guarantees the legitimacy of the website. Proper encryption is another consideration. This guarantees privacy. For port 443, TLS should be used. The TLS protocol should be used by client-server applications to secure data transfer confidentiality. Avoid using SSL because it is vulnerable to the POODLE attack, a man-in-the-middle exploit. Use AES 128/192, 256, or other encryption techniques when encrypting data. To ensure data integrity, hashing algorithms must be used. I advise employing either SHA-256 or SHA-3. It is not advisable to use MD5, SHA 0, or SHA 1. Attacks like birthday attacks and collision attacks are common against them. We must use efficient design, coding, and testing practices to stop zero-day assaults. A threat model offers a structure for implementing security activities, so using one is brilliant. The gaming room must use a SIEM (Security Information and Event Management) system to handle security warnings. Implementing IPS/IDS and firewalls is a clever idea. They could look at purchasing security as a service. To ensure safety, I advise employing vulnerability scanners. Offering honeypots and honeynets is another effective way to deceive a hostile

person. This can provide forensic details on how an evil user could take advantage of the system.