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	7/13/2022	Dante Trisciuzzi	Initial Proposal
1.1	7/26/2022	Dante Trisciuzzi	Platform Evaluation
1.2	7/10/2022	Dante Trisciuzzi	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

Draw It or Lose It is an android game built by The Gaming Room. The Gaming Room is looking to expand their platform targeting to allow the game to run on most commonly-available operating systems. We are looking to develop this game with a web development stack, giving us flexibility across multiple system environments from a single code base.

Design Constraints

The Gaming Room has provided us with some software requirements as such:

- ➔ 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.

In addition to these requirements, we also have some implied constraints:

- ➔ Strong server-client communication and security
- ➔ Cross-platform client

This final constraint, building a cross-platform application, would pose a challenge if we the client were looking to have a native application for each platform. The Gaming Room's choice to have a web-based version of the application makes this requirement trivial in implementation. We still need to consider that the application web page should have both mobile and desktop specific formats.

System Architecture View

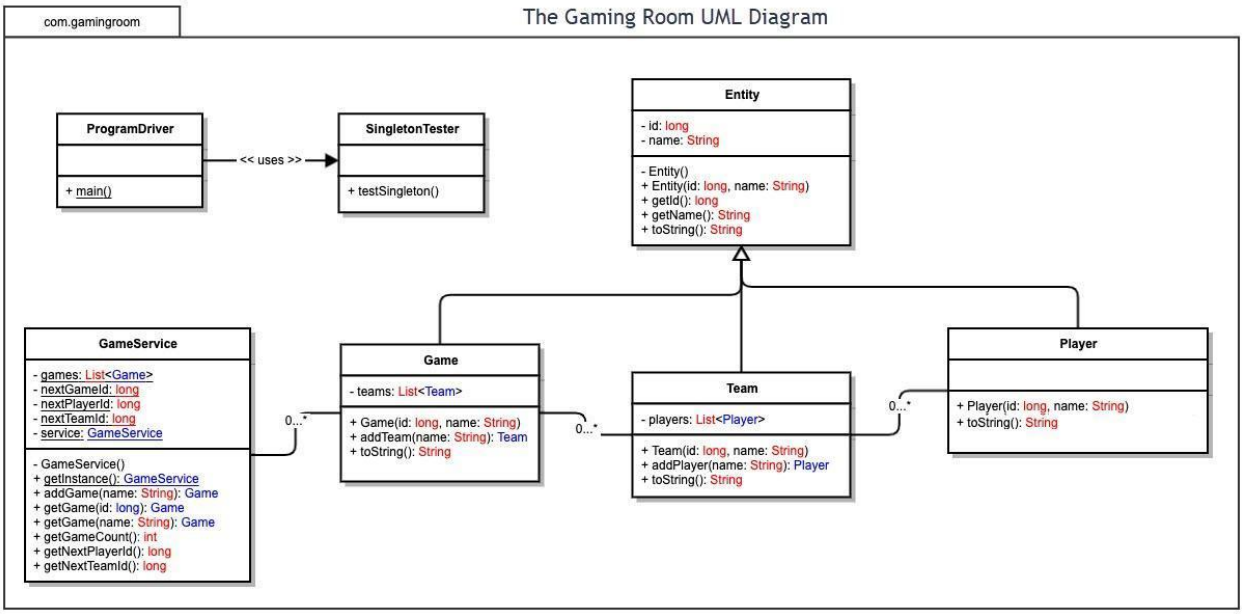
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

Domain Model

The com.gamingroom namespace contains all the classes necessary for the operation of the game client. Since we need to be able to index all the instances of the Game, Team, and Player classes with unique identifiers, we will use a base class (Entity) for all three to inherit from. This type of inheritance provides us with the ability to treat these classes in a polymorphic manner, since we can treat them all as an instance of the Entity class.

As we look at relationships between the GameService, Game, Team, and Player classes, observe that the relationship between them is cascading aggregation. This is to say that the GameService class can have any number of Game Instances, which can have any number of Team Instances, which can have any number of Player instances. Note that this is a one-way relationship, where a Game can have multiple teams, but a team does not have any Game.

We also have a ProgramDriver class that we use as our main entry point, and for implementing our SingletonTester class. The Driver class will implement the main server loop in production.



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	● Supports Java VM ● Accessible Shell (ZSH) ● Not cost effective ● Silicon Chips have non-standard architecture ● MongoDB Support	● Simple file system ● Cheap ● Easy to secure ● Accessible Shell (Bash Terminal Emulator) ● OpenLADP support ● MongoDB Support	● Best GUI Support ● More expensive ● Least secure ● Complicated Shell language (Powershell) ● Azure Cloud-based Active Directory ● MongoDB Support	● Unsuited for server hosting ● Least resources
Client Side (Middle Tier, web application)	● Additional work may be required for Mac standard browser support (Safari)	● Standard browser support (Chromium based)	● Chromium browser support ● Microsoft Edge support may add complexity to web applications.	● Chromium, iOS, Edge all are present on mobile devices ● Mobile-Specific layout will be necessary ● Proper UI scaling is key for mobile devices of various sizes, shapes and resolutions.

Development Tools	● Fully supported by Jetbrains' product line (Intel + Silicon) ● Silicon supported by VSCode for C/C++/Swift ● Client/middle Development with Java/JS/HTML/CSS ● Full support for .NET making ASP.NET or avalonia a viable option	● Many distros supported by Jetbrains' product line ● Support for VSCode ● Client/middle Development with Java/JS/HTML/CSS ● Full support for .NET making ASP.NET or avalonia a viable option	● Fully supported by Jetbrains' product line ● VSCode Supported ● Visual Studio Supported ● Client/middle Development with Java/JS/HTML/CSS ● Native support for .NET making ASP.NET or avalonia a viable option	● Development from mobile devices not recommended, because of ergonomic/QOL limitations

Recommendations

1. **Operating Platform**: I recommend that we target the linux OS when developing the server-side application. In particular, we should build and test our application on the same operating system that our servers will run. The current stable release of Ubuntu Server is an excellent choice, since it is by far the most common, and very well documented. Linux provides us with several important benefits for server hosting; less overhead taken up by unused services, simplicity of deployment, clear and easy authentication and security.

2. **Operating Systems Architectures**: Ubuntu Server has a wide range of support for different architectures. From the most common x86/x64 and ARM machines, to some more obscure enterprise solutions like IBM-Z and several cloud architectures (POWER, etc) and other little endian architectures. If we choose .NET as our primary runtime for the server, we can easily deploy to x86/64 and ARM machines as they are fully supported. POWER support for .NET is said to be coming soon.

3. **Storage Management**: Deploying the server on Linux remotely lets us use the extremely simple file system. We can set our permissions to allow only the specified user(s) access to even view the folder that contains our application and data. We can take advantage of the open source OpenLADP for active directory queries.

4. **Memory Management**: As with storage management, Linux stores runtime memory in a simple scheme that is protected the same way the file system is. Only the users with permissions can access the memory. Linux will launch our program by allocating a contiguous block of memory. Adding more games, teams, and players will increase the system memory usage.

5. **Distributed Systems and Networks**: Clients will communicate with one another via the server. The Draw It or Lose It client will communicate its turn to the server, and the server distributes it to the other clients in the game. I recommend a MongoDB database for keeping track of historical games. This gives us the ability to compress and back up games periodically for later access. Applying the 150% networking rule would be a good choice for this application, such that we should deploy servers to support 150% of the traffic we expect to receive. It's also worth noting that significant bandwidth will be required since the games will involve users sending and receiving images throughout. In terms of connectivity, servers should be hosted in various locations across the countries where the game will be available. This lets the client choose the server with the best connectivity (lowest ping response time), as well as protects the users from outages since we have server repetition.

6. **Security**: Security starts at the code level, beginning with secure coding practices, and extending through extensive unit and penetration testing. As for the security of user data when playing the game, we should use salted and hashed passwords (one-way encryption). In addition, we should allow the user to choose a display name, and this would be the only information distributed to other clients. In terms of server security, proper setup and maintenance of a Linux server will keep most attacks at bay, however, we should factor in the cost of proper red-team testing on the machines. Protecting user data is important, so only information necessary for gameplay should be stored on the servers and in the database. We also should make sure that database calls are only allowed from the local (server) machine. CloudFlare firewall protection for the game server is recommended.