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	11/12/2023	Matt Jackson	Initial Document
2.0	11/22/2023	Matt Jackson	Evaluate the characteristics, advantages, and weaknesses of various platforms
3.0	10/10/2023	Matt Jackson	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

The Gaming Room is developing an application called Draw It or Lose It. The game is currently only available on Android, but the team wants to expand availability through a web-based version. As such, consideration must be made to multiple user environments from desktop to mobile when designing the user interface and support functionality. Multiple teams with multiple players can participate in a single game, but only one instance of any given game will exist at one time.

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. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

Design Constraints

The web-based nature of this game assumes players will have consistent internet access. There needs to be a pulse taken from the users to ensure they are still a part of the game. Asynchronous play (not requiring players to go in a particular order) will allow the game to gracefully continue if the heartbeat test fails. The timer will also ensure that if no input from an entire team is received, the other team can continue with the game. The same nature requires the game to work on a number of platforms from mobile to desktop. Multiple browsers must be tested.

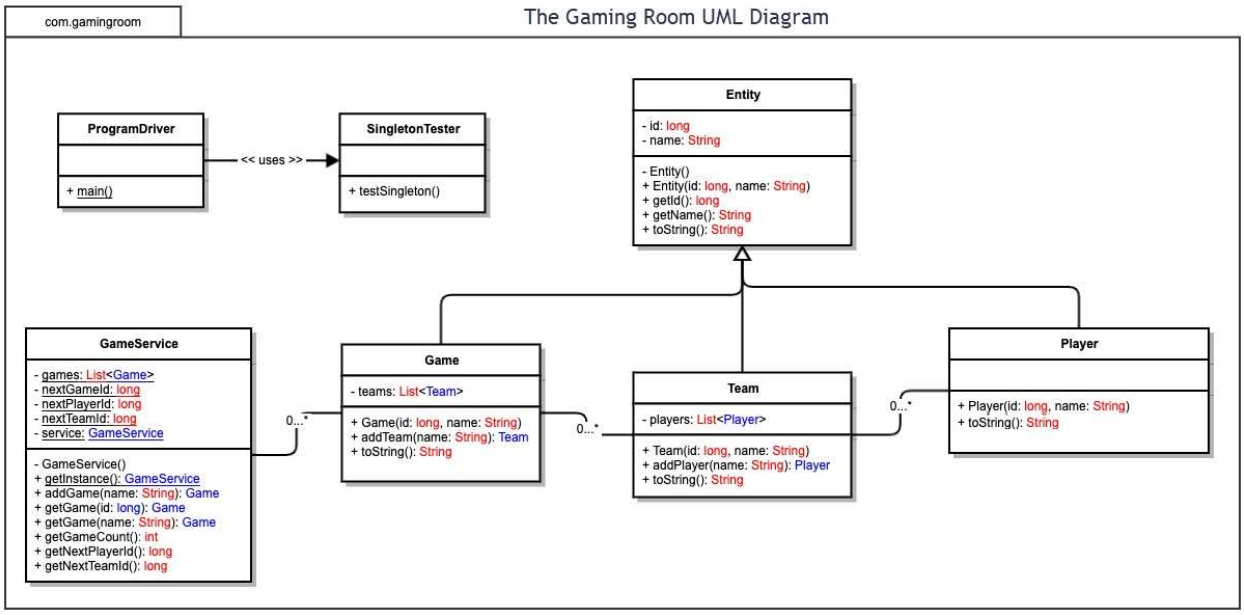
System Architecture View

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 abstract Entity class is the foundation for the Game, Team and Player classes. Each of these object types inherits a number of attributes and methods including a unique identifier that helps to ensure only one instance of a game or team exists at one time. In addition to the private id, each instance has a private name both of which encapsulate the data so that it is secured from the rest of the program. There are public methods to get the id and name, as well as one to display the object as a string. This toString method is overridden in each instance class. This is an example of the polymorphism principle of object-oriented programming which allows each object to have its own method of displaying information about the separate object types. The entities are constructed with an id and a name. The Team class has a list of zero or more players, and the ability to add a player to the list. Similarly, the Game class has a list of zero or more teams and the ability to add them. The game class will ensure that when teams are added, they will have unique names as well as Ids.

The Game Service class has one or more games associated with it in a private list. It also keeps track of the next gameId, playerId and teamId each of which are incremented when an object of that type is constructed. The game service is a singleton instance that will control the creation of the other objects to ensure games have unique names. It can identify a game by name or id, return the number of active games and the next player and team ids.



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 can run server-side software; hardware to support is expensive; not much third-party software	Most flavors are free, well documented, many open-source applications to choose from; may lack long term support	Common OS for servers and well supported, but it is proprietary and a popular vector for attack. Many third-party applications	Mobile devices may not have constant access to network, which is desired for a server to allow client access at any time
Client Side	Access to native hardware may require custom drivers. Developing with backward compatibility in mind can be challenging. Cost to publish to app store is $99 per year for an individual and $299 a year enterprise.	Can submit application to become developer for distro like Debian or host repository yourself for the cost of webhosting. Can develop for multiple distributions of Linux at once	Cost to publish to app store is a one time fee of $19 for an individual or $99 for a company. Backwards compatibility and multi-OS support is more easier than for a mac.	Can develop as native programs, cross-platform, or web-based. Native runs better, but is costlier to maintain and requires different code bases. Cross platform is easier to maintain but has less performance. Web-based has same code for web and mobile apps, but depend on the browser and have limited access to native hardware
Development Tools	Client development requires Xcode IDE which runs on Apple devices. Free with free developer account. Swift programming language / SwiftUI recommended. Learning curve no greater than any other basic programming language.	Many free IDEs available that support many programming languages. Many free plugin / dependency libraries available. Dependency libraries may not be maintained by creators.	Windows App SDK requires Visual Studio 2022 to develop for all version of Windows App SKD. Cross platform development available through React Native and .NET MAUI.	Many IDEs available, often free for open-source development. May have a subscription fee for proprietary development. Many programing languages used, development for iOS and Android most common with objective C and Java

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**: Each of the three major operating systems is a viable solution for the Gaming Room's development needs. Because a primary goal is to port Draw It or Lose it to different computing environments, virtual machines running different OSs could be used for native development and testing. All three support these. Linux is generally opensource freeware, but there can be a learning curve with hit-or-miss documentation and limited support for peripherals. Mac hardware tends to be more expensive. Windows offers a good middle ground by running on relatively inexpensive hardware and having good support and documentation. Windows also comes with its own VM software called Hyper-V.

2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>

3. **Storage Management**: Individual games would not need to be stored beyond their runtime, perhaps statistics could be kept for teams or individuals. Storage would primarily be used for backend development and hosting. Because multiple developers may need to access the same pieces of code, a Storage-Area Network system would be most appropriate.

4. **Memory Management**: A successful application will have many instantiations of the game running at any given time. A large amount of RAM would be needed for multiple instances to run simultaneously. One image would be loaded from storage into memory at the beginning of the game for each game in play.

5. **Distributed Systems and Networks**: Native applications could be created for each end platform (iOS, Android, Windows, Mac, Linux, etc.) or a language like Java could be used, placing the burden of individual implementation on the end user to have the Java Runtime Environment installed on their given platform. But the easiest way would be to develop the game as a web application and test it on multiple browsers. The burden of individual implementation would still be placed on the end user, but browsers are more ubiquitous and follow a very standardized method of rendering data.

6. **Security**: User accounts would be protected with a strong password and pre-selected challenge questions to reset their passwords remotely should they be forgotten. Company networks would be housed on a private network with only certain workstations and user accounts having access to the public servers located in a DMZ. The DMZ would have a stateful firewall preventing internet traffic from entering the private network if the connection didn't originate from the inside. A firewall would also be placed between the public facing servers and the internet to allow only game traffic in, prevent blacklisted IP traffic and gracefully handle any traffic surges that may overwhelm the servers.