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	May 18, 2022	Shayne Rushton	Vers 1.0

Executive Summary

CTS would like to expand their Android only game Draw It or Lose It to a web based application. Working with in house developers to create a web based application instead of creating one app for each available operating system, the goal Is to provide the game to the widest possible audience, which will help increase revenue through ads and in game purchases.

Design Constraints

Technical
- Server
  - To make the game accessible via web on multiple systems, a server is required, either a hosting service to serve the site, or a dedicated server maintained by CTS.

Personnel
- Staffing
  - Employees need to have a specific skill set capable of creating web applications

Operational
- Game should have the ability to allow one or more teams to play
- Each team Should allow multiple players
- Game and team names must be unique
- Only one instance of a game can be active in memory at any time

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.

Game play will be the same with the same interface.

Domain Model

The Program Driver class will contain the Main class and will use a Singleton class to test the Singleton design pattern.

The Entity class will be the main class from which the Game, Team and Player are composed. It will hold the ID and Name of each player and each Team since the Entity is comprised of an ID and a Name with methods to return all of the above.
A player can be on a team, but doesn't have to be on a team whereas a team must have players and a game must have teams, but a team doesn't have to play a game. Once a game is created it will exist in a list in the GameService instance as will each player and team created.

The Entity class should contain all common attributes and behaviors for each sub class to utilize. In this case, ID and Name are common to game, team and player as is the toString() method which prints a message to the screen on System.out subjective to the Object created along with it's instance variables id and name.

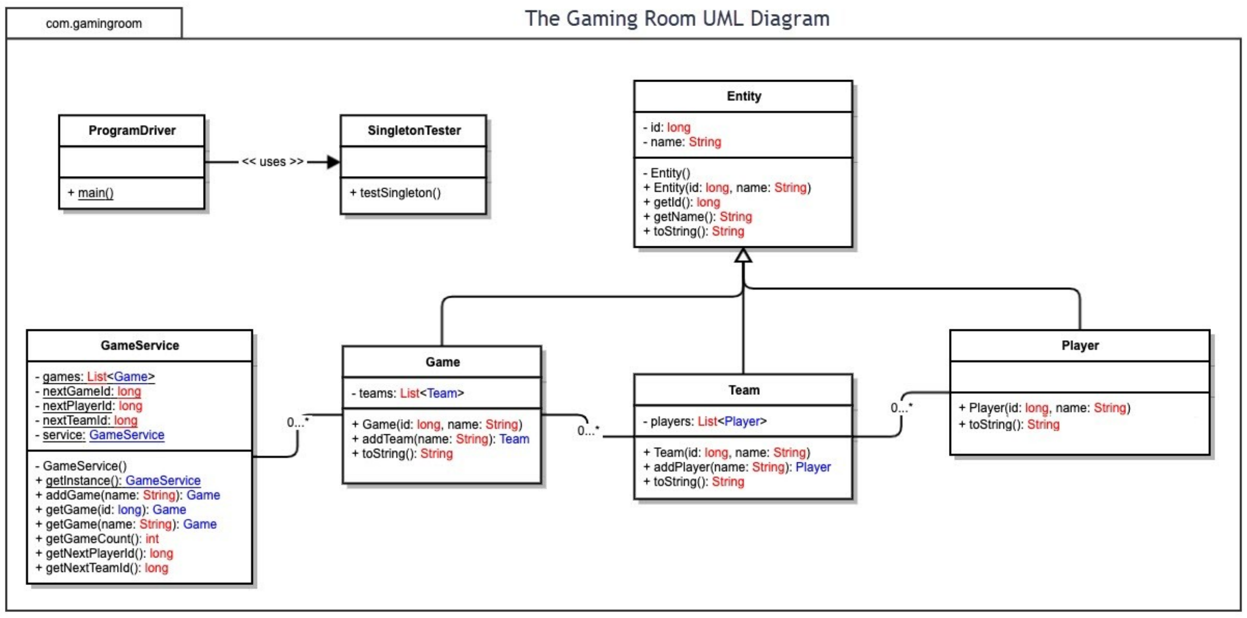
The GameService class will be used to hold the instance of the games created, keeping the Team and Player information for each game and delivering the game names and IDs along with verifying that game IDs and Names are unique. It also serves to deliver the next team and player to keep game flow correct.

Players can have the same name, but unique ids, however, team and game names and IDs must be unique. When a player is created, it calls the Player constructor which in turn calls on the super constructor, thereby abstracting the Entity constructor. The Team class behaves the same way.

Using the Entity inheritance and OOP principles, the Game class utilizes the Team class to create a list to hold teams and the Team class uses the Player class to create a list of Players. All are accessed via the entity class through accessor methods held within entity.

Polymorphism is shown through the accessor methods and the toString method which returns the requested value based on the object passed to it.

Encapsulation is accomplished by making the Entity variables and default constructor private, thus requiring the sub classes to use the super keyword for constructing a new object of that type.



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	Strengths: Networking Weaknesses Not good for large numbers of users,	Strengths: Ubiquitous as servers go, reliable, well supported, easy interface Weaknesses: many different versions	Strengths: offers deployment services, cost effective, well supported. Weaknesses: Interface	Strengths: Compact Weaknesses: storage capacity, processor capabilities
Client Side	In terms of mobile apps for Apple, quality standards are rigorous. Developers	Linux has no real market for mobile devices so a web app should suffice for this OS,	For windows, the most obvious choice is the .NET framework and C#, so	A great diversity among developers will be good here. The various operating

Development Requirements	**Mac**	**Linux**	**Windows**	**Mobile Devices**
	should know either swift or Objective C or C#. Otherwise, your basic web languages, such as JavaScript and PHP for a browser based client is fine	but for desktop applications, Java or C++ and possibly python should be considered.	developers should know these. Windows is reasonably popular so finding people to develop desktop and web apps for this platform should be easy	systems mean a little extra effort to create the apps required to run clients. Due to screen size, the desktop app is a better idea than a browser based app due to browser compatibility.
Development Tools	Developing with iOS should be done on a mac OS on xCode, and IDE for Mac and using the Swift language.	Linux app can be created on a variety of frameworks, including GNOME, KDE and Elementary OS. GNOME allows popular languages such as C, C++ and Python along with JavaScript	Windows offers a powerful and free IDE for community developers, but also offers one for purchase with all features. Visual Studio allows programming windows based application using C++, Java, Visual Basic and many many more. Very diverse.	Xamarin is a popular IDE for mobile devices. It allows cross platform development in one language and using the same APIs and data structures. Certainly the recommended app for mobile desktop applications.

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 ubiquitous nature of the Linux platform for server side application is recommended. For client side, the recommendation is to use the operating system's native application development and create a desktop application to handle client side operations. Since there is already an Android application, all that should be required is a Mac client. Even though a web based game would not present many challenges, it is also less common to see games on a browser that are worth playing according to https://www.juegostudio.com/blog/which-is-best-choice-for-gaming-a-mobile-browser-or-mobile-app.

   Recommend Linux driven server side with a Java based client side

2. **Operating Systems Architectures**: Linux is highly secure. It supports multi-threading and generally offers the best functionality over all. The hardware support is more general, meaning more options for physical updates. The open source nature of Linux also means that there will be much greater developer support in terms of updating and these updates can be quicker and more frequent. Cost is lower with Linux since it is open source and free to all. Most servers run Linux already, so in terms of delivering the best game service this is operating platform works best. It communicates well with Mac, Windows and Android.

   Recommend Unix based systems for this application.

3. **Storage Management**: Ideally, managing storage would use direct access with indexing so that each game can have its own set of images with an index pointing to each image used. This way, the images only have to be stored in one place and the index can deliver the one it points to for that game. This will eliminate storing duplicate images so space will be less. This might cause a slight performance issue in high user volume times, but should be negligible.

   Recommend direct access with indexing for storage management.

4. **Memory Management**: This game can be front loaded per game to each client. In doing so, direct access to memory should work. Once downloaded on the client, the client can handle the memory. Each game should be only a small amount of memory usage with a brief delay while the game downloads, then memory will be freed up before the next game is played.

   Recommend direct access with memory management also. Let the local machine handle the access to memory from game start to end.

5. **Distributed Systems and Networks**: With a distributed system, you have computers connected in another location, and each system has a scalability factor that allows more or less storage and memory depending upon the amount of user requests. This allows the server side to manage few or many requests without losing performance. The limitations of this system is that a client must have access to the internet. Down time on the distributed systems is very low. According to AWS, up time is guaranteed to be 99.99% with 4.38 minutes of permitted downtime per month (https://www.logicata.com/blog/aws-service-level-agreement/). The maintenance of a distributed system is also a cost savings, since any hardware upgrades or replacements will be the responsibility of the provider. This also means the most updated hardware is likely being used. Overall the distributed system will give the best performance, offer cost savings, and even allow programming in one language for the server side code, and since Java can also run on any platform, Java can be used to do all client side application development.

   Recommend Google Cloud Platform

6. **Security**: Using API to do authentication, along with two factor authentication at sign in will create a very secure and up to date app. With certain devices having a bio-metric sensor, we get better security still, but this is user driven and not something we can control. However, with Single Sign on and Two Factor authentication available, we can offer the most security for our users to protect their data and the servers.

   Recommend using two factor authentication and single sign on methods and API calls for authentication from client to server to ensure unique requests each time a login is done. In addition, detailed logging, verifying third party dependencies and strict validation of all input.