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	07/12/2021	Logan Parker	

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 a multiplayer game that shows team of players images, and the team that can guesses the item or phrase first wins. The game is currently available on android, however A web-based gamed that serves multiple platforms will be implemented.

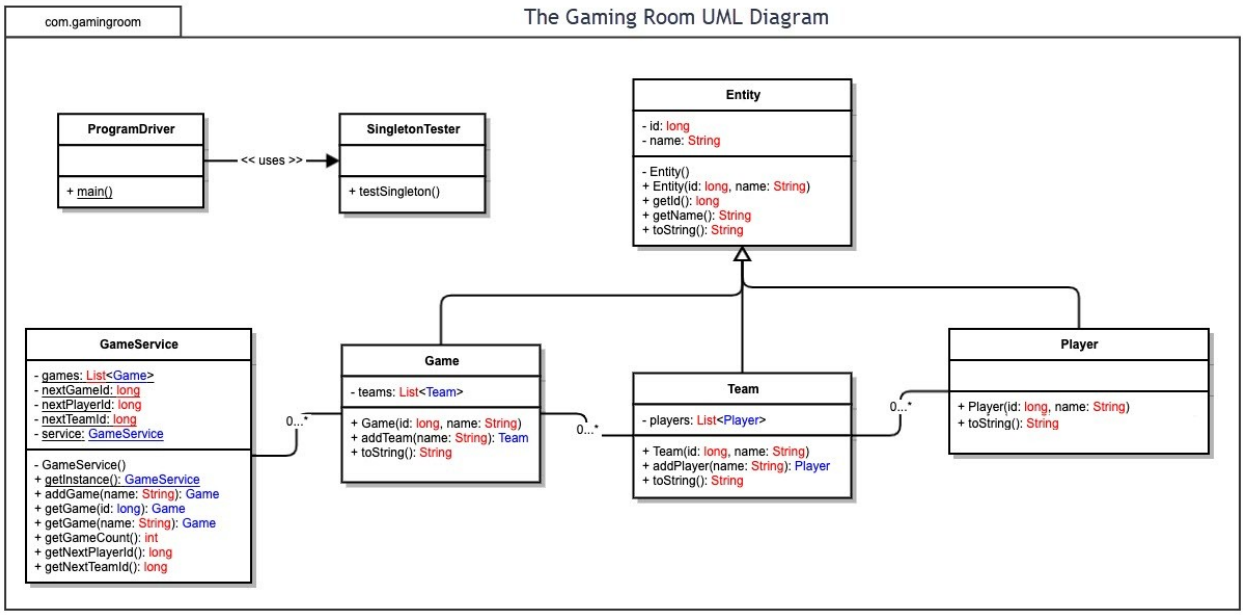
Design Constraints

- Must be available to a large variety of platforms
- Only one instance of the game should be able to exist at one time
- Game and team names must be unique
- Each team can have one or more players
- Each game must have one or more teams

Domain Model

This is the class diagram for the game as a whole. The program driver is the main class that kicks the program off, and uses the SingletonTester class to test that only a single instance is running. The Entity Class is a Parent object to the Game, Team, and Player Classes. It contains basic and broadly used components. The GameService class contains most of the core logic of the program, and executes much of the process that move the game forward. It is responsible for instantiating the Game class and managing the Teams and Players. The Team class contains the many possible Players in a team, and the player Class holds the unique player Id.

This UML diagram shows 4 of the major object-oriented principles. It shows encapsulation by keeping logic and data boxed into their own areas, e.g., Teams containing internal team logic and data. It shows abstraction by again boxing off logic, so that the developer doesn't have to dive too deep into the details to understand higher level functionality. It uses inheritance when Team, Player, and Game all inherit the Entity class. Finally it uses polymorphism, by having classes with various amounts and types of data, with somewhat different functionality, although this is a small example.



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	- comparatively slow - license nightmare - constant unnecessary but forced updates - user friendly sometimes	- fast - free (for os) - Open Source (can be custom tailored) - very scalable - takes a little more knowledge	- Somewhat simple to use - Most people are familiar with Windows - Costs money - Scalable	- A mobile OS should not be used on a server...
Client Side	- Would take some specific Mac knowledge, or research time - Most of the application would be the same on Widows, Mac and Linux - Would need often and unnecessary updates	- Would need to be created many different ways for many different distributions - Most of the Application would be the same	- Windows is the platform for most games - Would need to be occasionally updated	- Would need to be created twice for apple and android, or utilize a platform like flutter or react native.
Development Tools	- Eclipse is good for Java development - Most modern games are built in C++ or C# for speed. In which case I would use Visual Studio - .NET	- VS Code with extensions - eclipse - game libraries - Java / C# - .NET	- VS Code with extensions - Eclipse - various libraries - .NET	- For non-native development use Flutter/Dart or React Native using Typescript - Android Studio / Java - swift

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**: Use a EC2 auto scaling instance on AWS using an Ubuntu OS.

2. **Operating Systems Architectures**: AWS allows you to run servers that scale very well. Additionally, Ubuntu is fast, reliable, easy to control, and will scale very will to the demands on the server. This will create an API that links various instance and platforms together.

3. **Storage Management**: For long term storage I would use a AWS relational Database using MySQL. This will allow the storage of users and teams as well as numbers for user retention and data regarding how players are using the software. AWS regularly saves snapshots making backing up the database very easy.

4. **Memory Management**: Local storage will come down to the platform that is running on the client.

5. **Distributed Systems and Networks**: Creating a RESTful API will allow any client platform to connect with it. The use of an API means that the language and platform are irrelevant as long as the program can send JSON requests to the server. It may be necessary to have multiple servers running at the same time to avoid performance issues and outages.

6. **Security**: Users should have to sign in to an account. The server and database should be secured with two-factor authentication. All connections to the API should utilize a SSL connection. Passwords should be stored encrypted. All of these features are very easy to accomplish using modern API tools, AWS, and Linux tools.