Win, Lose or Draw

CS 230 Project Software Design Template

Version 1.0

Table of Contents

CS 230 Project Software Design Template 1
- **Table of Contents** 2
- **Document Revision History** 2
- **Executive Summary** 3
- **Design Constraints** 3
- **System Architecture View** 3
- **Domain Model** 3
- **Evaluation** 4
- **Recommendations** 6

Document Revision History

Version	Date	Author	Comments
1.0	03/19/2022	Vincent Snow	Executive Summary, Design Constraints, and Domain Model sections completed
1.0	04/02/2022	Vincent Snow	Added Evaluation table
1.0	04/14/2022	Vincent Snow	Completed Recommendations

Executive Summary

The Gaming Room wishes to have a web-based application version of their mobile app game 'Win, Lose or Draw' developed that serves multiple platforms, and can be played by multiple players and teams on the same game instance.

Design Constraints

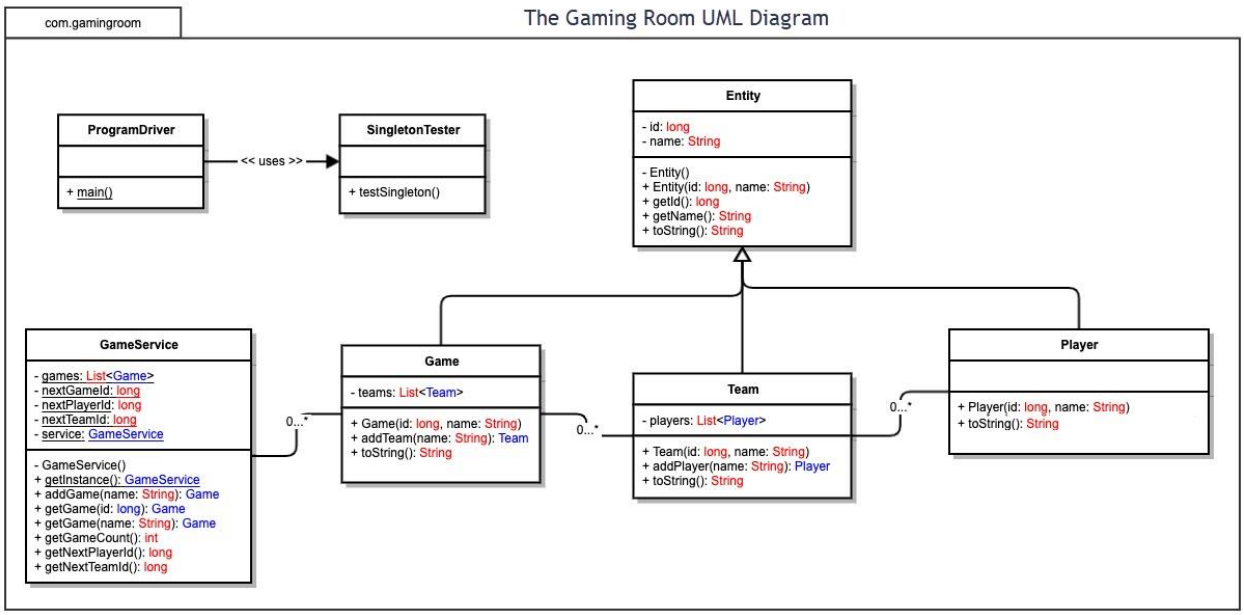
-Games must have the ability to involve one or more teams.
-Each Team must be able to include multiple players.
-Game and Team names must be unique to allow a check to prevent duplicates.
-Only one instance of a particular game can exist in memory at a time.
-Games render an image over 30 seconds from a library of stock images, during which time teams can guess the puzzle.
-If the Team does not guess within 30 seconds, other teams have one chance to guess within 15 seconds.
-A Game consists of four rounds of 1 minute length.

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 GameService class is a singleton class that wraps the entire program and spawns the game instances with its addGame() method, which creates a unique Game and adds it to a list of active games. It uses a private instance of itself to refer to and check to be sure that only one of it exists. The encapsulated fields 'next____Id' are used to generate unique ID numbers that ensure all Games, Players, and Teams are unique objects and exist only once. 'Entity' is a parent class from which Game, Team, and Player inherit an id and name field as well as its related methods. Because of polymorphism, these three classes can be interchangeably required as an argument or referred to simply as 'Entity' type. A GameService can contain multiple unique Game instances, which can contain multiple unique Teams, which can contain multiple unique Players.



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	+ More secure than others. + The URLs are not case-sensitive. +Has easy-to-use tools. - More expensive than others. - Limited tools and software.	+ Cheapest option + Compatible with many other OS software + Best performance, scalable system that manages memory well	+ Wide array of popular tools and software. + Only server compatible with Visual Basic and .Net + Continuous security updates	-Unlikely to have adequate storage space for hosting a web application with a high number of users and database. -Limited software is also an issue.

Client Side	Supports both Chrome and Firefox browsers to view HTML web applications. Everything needed to view websites or run GUI applications is pre-installed.	Supports both Chrome and Firefox browsers to view HTML web applications. Users can install desktop environments for certain programs, and Wine to run Windows programs on the system.	Supports both Chrome and Firefox browsers to view HTML web applications. Everything needed to view websites or run GUI applications is pre-installed.	Supports Chrome and Firefox browsers to view HTML web applications. Everything needed to view websites or run GUI applications is pre-installed.
Development Tools	XCode IDE can be used to write/edit in Objective-C to create native Mac applications. Our team would need to learn the software and purchase Mac PCs to use it.	Glade IDE can be used to create GUI programs in Java and other languages. We could save time and money by simply creating a Windows app, which Linux users can run using Wine.	Since we are using a Maven project in Eclipse written in Java, we can simply use the launch4j plugin to wrap the .jar in an .exe file with the JRE if we want to create a GUI desktop application. It can also publish a web page, as we have done for testing, if we want to make it run in the browser.	Visual Studio or Android Studio can be used to create mobile apps for Android. They can use C++, Java and many other languages. Xcode on Mac can be used to create mobile apps for iOS. Time is needed to create a different design, whether GUI or web application, to suit the screen resolution of mobile devices.

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**: I recommend that we use a Linux server for hosting the application. It is compatible with much common software used on Windows and will be the least expensive choice. We have no need to spend extra funds on a Windows or Mac server for extra security and updates, and we will not be using .NET or Visual Basic in this project. Linux also offers superior performance and memory management, which will be important as the player base grows.

2. **Operating Systems Architectures**: We should use a simple, trustworthy client-server architecture. This would save money as opposed to using cloud storage as well as be safer and more reliable. It is unlikely that we will need to use a cluster of servers, as the game data and resources are not expected to expand much, but more can be added later if necessary. The game's main code and downloadable files will be held on our server, along with a database of the images and player credentials. Necessary files to run the game can be installed on the client's device, such as game objects, a copy of player credentials, and current image in the game.

3. **Storage Management**: A database held on the server will hold images and user credentials, and it must be regularly maintained to delete unused accounts beyond a certain age and check for unusual activity. The code itself should contain prevention measures and validation to keep copies of the same user and offer a way for users to recover an account if the password is forgotten to prevent creation of new accounts in that case while old, unused ones take up space. Because we are writing the main program in Java, we will have the benefit of Java's automatic garbage collection to prevent memory leaks on the server as the program runs.

4. **Memory Management**: The app should be coded to delete unneeded files (game objects, images, user credentials) held in a folder in RAM when the game is closed to prevent overloading the client's device. This way, even if the app is closed improperly, it can still clean the folder the next time it is properly shut down, or the memory will be cleared when the device is turned off. All temporary resources needed to operate should be kept in memory. The program should practice 'lazy loading' so that each image is not loaded from the server storage and held in memory until the moment it is needed. The images in storage should be optimized for different devices, so that the application on the client can fetch the best image for the device it is running on.

5. **Distributed Systems and Networks**: The server and clients should communicate through HTTP protocol messages that retrieve, create and update resources and data sent as JSON objects. This will enable the server to understand messages sent by many different clients such as mobile phones, gaming systems, and various personal computer OS that may have different requirements for what language their version of the app is written in. Any device with an internet connection and a compatible version of the application will be able to play the game using the same server with this method.

6. **Security**: All user passwords and personal data should be encrypted so that it cannot be read by a human when sent through JSON objects in the HTTP requests. This way, if intercepted, their information cannot be stolen by malicious attackers or leaked through an accident. The copy of user credentials on the client should be encrypted. Storing our data on a traditional server lessens the possible exposure when compared to using cloud storage. If the game will include microtransactions in the future, extra care must be taken to protect personal information such as using a safe 3rd-party handler and not saving any credit card information on the server. Lastly, we must protect the server and game from intentional and unintentional interruptions. To prevent overloading the server, all input from the user should be validated by checking for length and special characters before sending or storing data.