CS 230 Project One Game App

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

| Version | Date | Author | Comments |
|---|---|---|---|
| 2.0 | 11/26/2023 | Libby Felkay | Changed code provided to working code |

**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 wants to make the game, Draw it or Lose it web based. The game will need to be cross platform with client side software for each system. The game should be coded efficiently for all platforms to use.

**Requirements**

1. A game will have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. 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**

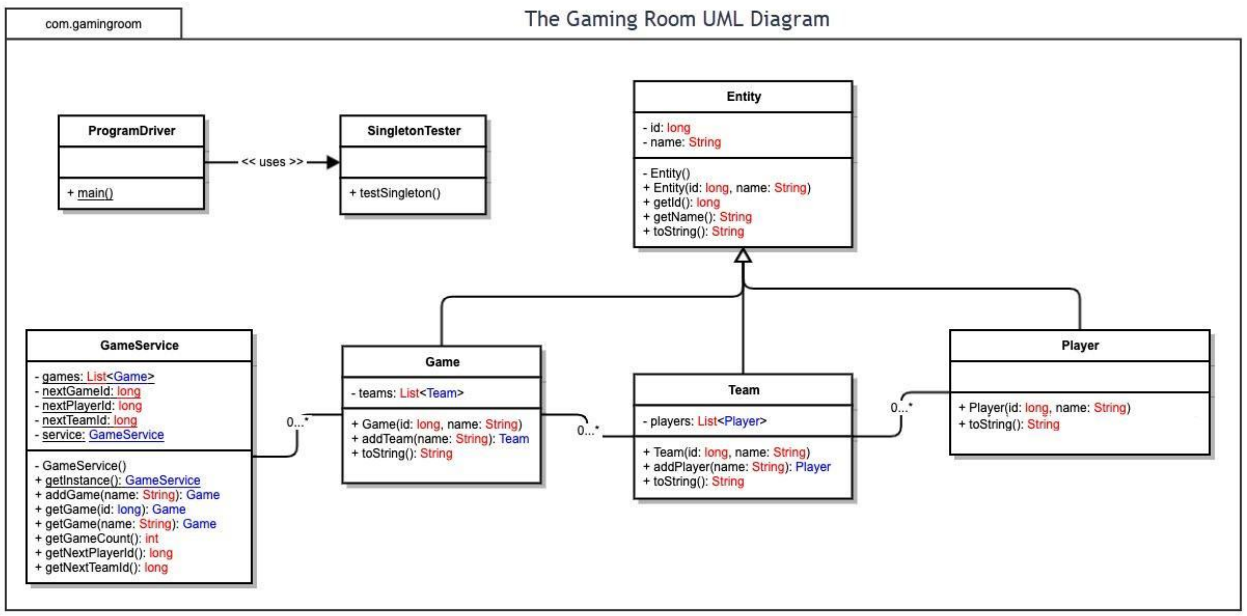
Since it is a cross platform web based game, the application must have server side data easily adapted for each client side OS. There will be different development kits for each OS. The game being cross platform means that the data will have to transfer between all OS seamlessly. With these in mind, we will have to focus on companion programs for each specific OS that all share a common server data type.

**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**

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>



**The OOP principles in the UML are abstraction, inheritance, polymorphism, encapsulation Encapsulation is used when each set of data is represented as public or private. Inheritance is used when classes game, team, and player inherit from entity class. Abstraction is shown by only having the important parts of the code represented in the UML. Polymorphism is shown when the classes can use variables stated in entity.**

<u>**Evaluation**</u>

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** | Like all Apple products, MacOS works really well with other Apple products. They all transfer data very well, a Mac based app would move to iOS quite well. Unfortunately, MacOS is not the most common OS. The benefits of sharing within the apple ecosystem are not really worth the other limitations. | Linux is free and open source. It is highly compatible with many web technologies such as databases and web servers. It is known to be stable and secure.It is the most efficient with resources. It has a strong community and plenty of documentation. It does have a significant learning curve and the graphics aren't nearly as polished as other OS. | WIndows is the most common OS for computers. The UI is very easy to use and it is compatible with almost everything. It has active directory integration to simplify authentication. Windows does use more resources than Linux and usually has a licensing cost. It also has a significant amount of updates required compared to others. | Mobile devices are portable and have many sensors and haptics to use. Mobile devices don't have nearly as strong hardware as computers do. They also use a lot of power, potentially overheating the device. |
| **Client Side** | Mac may be expensive as you need many products available for testing. Mac will also have less documentation than linux or windows so expertise would be needed. | Linux cost would be minimal if anything. Since there are many linux distributions, saying "linux compatible" means much more testing than just one windows version. You would likely want to collaborate with people in the Linux community. | WIndows development may have licensing fees. Testing and development are generally efficient and speedy. The integrated systems may be useful for development. Since Windows is most commonly targeted for breaches, security should be maintained. | App stores charge fees for apps released on them. You will need to have testing for the most common mobile OS like iOS and android. Many different screen sizes and ratios will need to be accounted for. Specific expertise in certain languages would be required. |

| | | | | |
|---|---|---|---|---|
| **Development Tools** | Swift and maybe objective c are the languages used. Xcode is the macOS IDE. Flutter can be used to make the code cross platform. | c,cpp,java,and python are all commonly used. Visual Studio, Eclipse, intelliJ are viable IDEs. Flutter can be used for cross platform. | C# is the primary, java and cpp are also used. Visual studio is microsoft's IDE. Xamarin can be used for cross platform as well as others. | SInce mobile can mean iOS or ANdroid, they both have different details. iOS uses swift, Android uses 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**: The OS I would recommend is linux. The focus is a web based game that is cross platform.

2. **Operating Systems Architectures**: Linux has several main architectures.
   x86 (32-bit and 64-bit): Commonly used for desktops and servers.
   ARM: Widely used in embedded systems, IoT devices, and mobile phones.
   PowerPC: Used in some servers and embedded systems.
   MIPS: Found in networking devices and embedded systems.

3. **Storage Management**: Identified Storage Management System:
   For Linux, the ext4 file system is commonly used for storage management. It offers features such as journaling, scalability, and support for large file systems. Additionally, tools like LVM (Logical Volume Manager) provide flexibility in managing storage volumes and snapshots.

4. **Memory Management**: Memory Management Techniques for Draw It or Lose It Software:
   Linux uses virtual memory and employs techniques like paging and segmentation for memory management. The kernel ensures efficient allocation, deallocation, and sharing of memory among different processes. The software should be optimized for effective memory usage, and developers should consider factors like caching, swapping, and avoiding memory leaks.

5. **Distributed Systems and Networks**: Communication between Various Platforms:
   To enable communication between various platforms, a distributed system approach can be employed. Technologies like RESTful APIs, WebSockets, or a message broker can facilitate communication. The software components on different platforms can interact through well-defined APIs. Dependency management should consider issues like network connectivity, potential outages, and error handling.

6. **Security**:

- Network Security: Use secure communication protocols to protect data during transmission.
- Access Control: Implement proper user authentication and authorization mechanisms.
- Data Encryption: Encrypt sensitive data both in transit and at rest.
- Code Security: Follow secure coding practices to prevent vulnerabilities.
- Regular Updates: Keep all software components and dependencies up to date to patch security vulnerabilities.