TALLER 5

Mateo Parra Ochoa

202213933

1. Descripción del proyecto y patrón

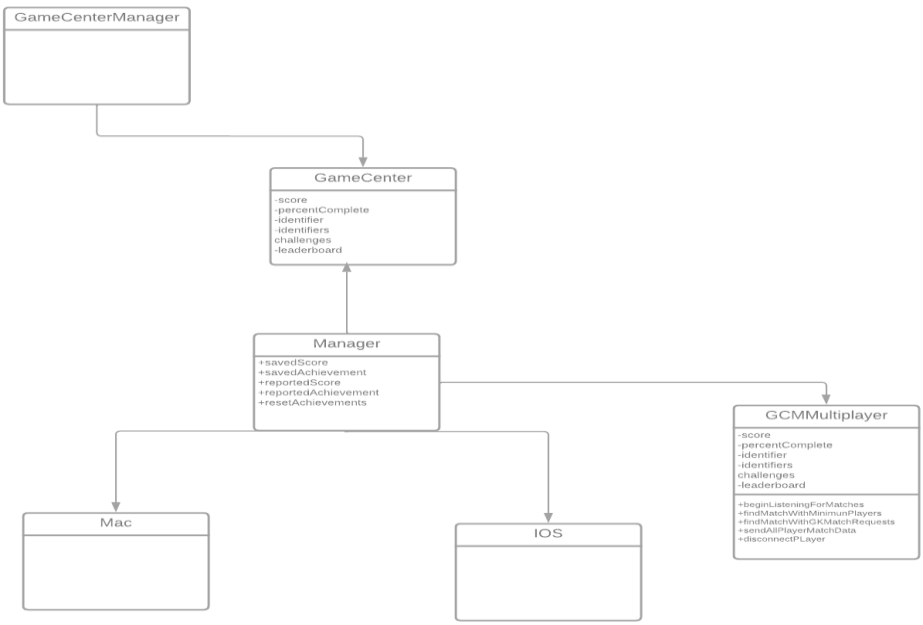
El patrón que se va a analizar en este documento es el patrón “Singleton”, un patrón de diseño creacional que garantiza la existencia de una única instancia de una clase y proporciona un punto de acceso global a esta instancia. Su objetivo principal es restringir la creación de objetos de una clase a un solo objeto y asegurar que no haya más de una instancia en todo el sistema. En términos simples, el patrón permite crear una clase en la que solo puede haber una instancia en ejecución, y proporciona un mecanismo para acceder a esa instancia desde cualquier parte del código. Esto es útil cuando se desea tener un objeto único que mantenga y controle ciertos recursos o datos compartidos en la aplicación.

Para analizar este patrón se escogió un proyecto de Git llamado “GameCenterManager”, GameCenterManager es una implementación específica de un administrador o gestor para interactuar con el Game Center desde una aplicación de iOS desarrollada en C++ y Swift. Esta clase proporciona una interfaz fácil de usar y encapsula la lógica necesaria para interactuar con el Game Center, como autenticación de jugadores, envío y obtención de puntuaciones, logros, entre otras funcionalidades.

URL: https://github.com/nihalahmed/GameCenterManager.git

2. Estructura del Diseño y Aplicación del Patrón

Esta implementación divide las funciones entre los métodos de las diferentes clases que hay dentro del proyecto, haciendo un análisis y extrayendo los componentes principales, decimos que un diagrama tipo UML acerca del diseño del GameCenterManager se ve así:



Este sería un prototipo aproximado de cómo se vería el diagrama de diseño del proyecto, sin embargo, no se asegura que este sea cien por ciento preciso y es altamente probable que se hayan ignorado algunas de las partes que lo componen, sin embargo, este diagrama es útil para poder hacer un análisis más acertado.

```
+ (GameCenterManager *)sharedManager {
    static GameCenterManager *singleton;

    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        singleton = [[self alloc] init];
    });

    return singleton;
}
```

```
+ (GCMMultiplayer *)defaultMultiplayerManager {
    static GCMMultiplayer *singleton;

    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        singleton = [[self alloc] init];
    });

    return singleton;
}
```

Estos son fragmentos cortos del código fuente en donde se implementa el patrón y se utiliza de manera directa sobre el proyecto y aunque estas imágenes solo son una parte reducida de la labor del patrón sobre el negocio, nos sirve de sobra para explicar cuál es la funcionalidad de este patrón sobre el GameCenterManager.

3.Aplicacion del Patrón

En un proyecto como el de GameCenter Manager es útil implementar Singleton pues el patrón garantiza que solo haya una única instancia de una clase en todo el programa. En el contexto real del proyecto esto podría significar que solo existe una instancia del administrador de Game Center en toda la aplicación y después de haber hecho una pequeña reconstrucción del diseño podemos decir que eso se cumple en cierto modo.

Una de las ventajas de singleton aplicado a este caso es que se puede acceder a la instancia en cualquier parte de la aplicación de manera global. Esto puede ser conveniente si necesitas interactuar con el Game Center desde múltiples componentes o clases de la aplicación sin tener que pasar explícitamente la instancia de "GameCenterManager" como parámetro, además de que todo el programa gira en torno al game center entonces es necesario instanciarlo múltiples veces y singleton hace mucho más simple y dinámico esta instancia.

Por otro lado, podemos hablar del costo en memoria de las operaciones de este proyecto pues singleton garantiza que solo haya una instancia de "GameCenterManager", lo que puede ayudar a controlar el uso de recursos y evitar duplicación innecesaria.