UML

# Introducción

En Unity lo común es crear un Script por cada función que deseas que haga cada objeto por lo que no da lugar a UML, pero con intención de aplicar a este proyecto más complejidad hemos decidido hacer una adaptación de lo que es el patrón de diseño fachada (facade).

Este patrón tiene el fin de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos. En Unity el resultado final de cualquier estructura ya es así pero como hemos dicho por hacer algo diferente y emplear más conocimientos hemos utilizado este patrón.

Nuestro diseño ha sido basado en los casos de uso que se diseñaron antes que este diagrama por lo que se mantiene en la medida de lo posible todos los nombres que se le dieron a las acciones de cada rol en su momento.

Bien, centrándonos en el diagrama en sí, tenemos todos los roles que se definieron en los casos de uso como interfaces de la fachada. Estos roles tienen como métodos aquellas acciones que puede ejecutar dicho rol. Estos tres roles implementan todos sus métodos en sus clases correspondientes, estas clases almacenan la mayor parte de código ya que luego a través de la interfaz action se dividen todos estos métodos en acciones independientes separadas cada una en una clase. Estos son los denominados actions y es la acción que implementara el objeto dentro del mundo de Unity.

Otro de los motivos por los que se ha escogido este patrón es la facilidad a la hora de meter nuevas funcionalidades. El tener una estructura tan lógica y tan humana es mucho más fácil de adaptar.

# Definiciones

A continuación, se hará una breve explicación del vocabulario que puede ser usado en el documento:

* Clase: Modelo que define una serie de variables y métodos apropiados para operar con dichas variables o las que reciben por parámetro.
* Objeto: Es una clase con atributos y métodos que se asemeja a la forma de pensar humana. Por ejemplo, clase coche, tiene atributo rueda, ventana, volante y caja cambio y de métodos pulsar embrague, pulsar freno, pulsar acelerador, cambiar marcha …
* Interfaz: Medio común a varios objetos no relacionado mediante la que se pueden comunicar.
* Método: Conjunto de instrucciones a las que se les asocia un nombre para poder usarlas en algún lugar del programa.
* Herencia: cuando una clase hereda de otra esta hereda consigo todos los métodos de la clase padre.
* Composición: Es una relación entre clases y se da cuando la clase A usa métodos de la clase B siendo la clase B interna a la A. Por ejemplo, un edificio tiene ventanas pues la clase A sería edificio y la clase B ventana. Las ventanas mueren si el edificio muere.
* Agregación: Relación entre clases que se da cuando la clase A usa métodos de la B una clase externa a la A. Por ejemplo, una empresa tiene clientes. Si la empresa muere los clientes siguen vivos.
* Asociación: La clase A depende de la clase B, la A esta asociada a la B, la A conoce la existencia de la B, pero la B de la A no. Por ejemplo, tenemos un taxi y un chofer. El taxi necesita ser conducido por un chofer, luego necesita acceder a sus métodos, pero el chofer no necesita al taxi.
* Dependencia: Es la relación más débil de todas se da cuando una clase usa atributos o métodos de otra sin más. Es la relación más básica.

# IAction

Tiene un solo método que implementaran todos los action del programa, este método es execute. Dentro del método execute todos los actions harán llamada a la acción correspondiente definida en las clases contenedoras de todo el código de cada rol.

# GameManager

Es la única clase en este proyecto que podemos decir que pertenece a la lógica, esta es una contenedora de objetos y variables comunes a varias partes del programa o simplemente necesarias para recordar algún valor permanente. Por tanto, esta clase tiene varias dependencias con bastantes actions.

# IninioNoLogueadoFacadeDelegate

Interfaz que hereda de IconectableFacadeDelegate. Esta interfaz contiene todas las posibles acciones que debe realizar el rol de niño no logueado. Estas acciones son: Solicitar inicio de sesión, confirmar inicio de sesión, solicitar recordar sesión, confirmar registro, solicitar registro y volver.

Todas esas acciones nombradas son métodos que debe tener obligatoriamente la clase que implemente esta interfaz.

# NinioNoLogueadoFacadeDelegate

Clase que implementa la interfaz INinioNoLogueadoFacadeDelegate, por tanto, esta está obligada a implementar todas las acciones que la definían. Esta clase es la que lleva todo el bloque de código referente al rol del niño no logueado, estas acciones después serán llamadas por los actions. Para codificar estos métodos en esta clase se han necesitado más lógica que se encuentra estructurada en los casos de uso.

# Actions de NinioNoLogueadoFacadeDelegate

Estos se encargan de llamar al método que necesiten de NinioNoLogueadoFacadeDelegate. Las clases action para este “rol” son las siguientes:

* SolicitarInicioSesionAction: Se ejecuta cuando el usuario pulsa sobre el botón que conduce a la escena de iniciar sesión. Este botón se encuentra en la escena principal.
* SolicitarRegistroAction: Este se ejecuta cuando el usuario pulsa sobre el botón que conduce a la escena de iniciar sesión. Este botón se encuentra en la escena principal.
* VolverAtrasAction: Este método lleva a la escena principal al niño. Los botones que ejecutan esta sentencia están en la escena de registro y de inicio de sesión.
* ConfirmarInicioSesionAction: Se lanza cuando el usuario pulsa el botón de iniciar sesión en la escena de inicio de sesión. Este puede que de lugar a mostrar un error o que nos cargue la escena lobby.
* SolicitarRecordarSesiónAction: Se ejecuta cuando el usuario pulsa el botón iniciar sesión en la escena de inicio de sesión con la casilla de recordar seleccionada.
* ConfirmarRegistroAction: Este se lanza al pulsar el botón de registro en la escena registro. Puede dar lugar a mostrar error o a completar el registro correctamente.