Patrones de diseño

Para esta Práctica se emplearán diversos patrones de diseño, entre ellos: Singleton, Factories, Bridge y Facade.

* Singleton:

Singleton se utiliza en la creación de Mario, pues en todo momento existirá un único Mario, pues es el personaje principal. En caso de aplicar un modelo multijugador en el futuro esta estructura no se verá comprometida, pues existirá una nueva clase Luigi y una clase padre de ambas jugador, por lo que seguirá existiendo un único Mario.

* Factories:

Las factorías son muy útiles en la creación de enemigos y de powerups, pues en ambos casos la cantidad de subclases es muy extensible, por lo que intentar gestionarlo todo desde la propia clase puede ser ineficiente. De este modo se crea la factoría de enemigos y de powerups de las que heredaran las factorías de cada enemigo y cada powerup

* Bridge:

Bridge es un patrón de diseño que se utiliza para separar la implementación de la abstracción. En el caso de este proyecto será útil para la

* Facade:

Facade se implementa en todos y cada uno de os managers, pues estos se encargaran de forma interna de toda la lógica y procedimientos necesarios, simplificando en gran medida la aplicación de dicha lógica.

* Flyweight

Este patrón de diseño se utiliza en gran medida en la creación de enemigos y powerups, pues cada uno de ellos tendrán unas características comunes que se guardarán en la clase en vez de en la instancia. Esto reduce en gran medida el uso de memoria.

* Command:

Este patrón de diseño no queda reflejado directamente en el diagrama uml, pues el número de clases dependerá del número de acciones, pero se utilizará al definir la lógica del jugador, permitiendo una expansión a dos jugadores en el mismo ordenador reasignando las teclas empleadas con facilidad, así como permitiendo la implementación del imput buffering, una mecánica imprescindible en videojuegos como mario, donde la fluidez de los movimientos es esencial.

* State:

State se aplicará principalmente para el movimiento de Mario y para los power ups, aún que puede extenderse al movimiento de los enemigos en un futuro.

La idea es evitar la creación innecesaria de condicionales. Esto se ve claramente en la pérdida de los powerups, pues perder una flor te lleva a ser grande, perder el grande a ser pequeño, pero perder la estrella (por tiempo) te lleva a tu estado anterior, por lo que surgen una cantidad innecesaria de ifs que se solucionan fácilmente con una máquina de estados.