Polimorfismo

Un mensaje polimórfico en este trabajo (Carpincho mayhem) puede ser en el method agarrar(objeto) que aparece en la clase Personaje y los que se aprovechan de ese mensaje son las instancias de la clase personaje como pueden ser personaje y personaje2.

Texto

Descripción generada automáticamente

Lo que realiza este mensaje es que el personaje puede “agarrar” cualquier objeto que entienda el mensaje agarrar sin importar lo que este objeto haga por ejemplo el personaje puede agarrar tanto una munición, pasto o un mate. En este caso la munición le permite al personaje cargar el arma, y el pasto y el mate le suman vida. Personaje puede utilizar este mensaje sin importar lo que este haga.

*Efecto munición*

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

*Efecto pasto y mate*

Texto

Descripción generada automáticamente

Colección

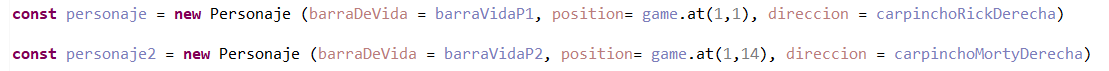
Las operaciones de colección que se utilizan son forEach y all. En el caso de la operación forEach es con efecto, tenemos una lista de supervillanos el cual le decimos a cada villano que se mueva, también tenemos el caso de game.addVisual que hará que cada villano aparezca en pantalla. En el caso del all es una operación sin efecto ya que al ejecutar la operación villanos.all({villano => villano.muerto()}) nos devuelve un True o False que nos dirá si todos los supervillanos están muertos, en este caso no realizara una “acción” si no retornara una Bool.

Imagen que contiene Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Clases

En el trabajo utilizamos diferentes clases una de ellas es Personaje, utilizamos esta clase ya que como tenemos dos personajes (personaje y personaje 2) que tienen el mismo comportamiento y para no tener código duplicado podemos definimos una clase Personaje donde esté definido el comportamiento de todo el personaje. Utilizamos la clase Personaje para no repetir lógica, estos dos personajes los instanciamos cuando necesitamos que diferentes objetos tengan el mismo comportamiento.



Herencia

En el trabajo se utiliza herencia cuando necesitamos utilizar algún método existente o atributo de una clase en específico, sin necesidad de repetir lógica, es decir, heredar su comportamiento y las propiedades del mismo. Utilizamos la herencia en este caso la utilizamos en villano y supervillanos para realizarla utilizamos a villano como super clase y supervillano la heredara. Como Supervillano no utiliza todos los mensajes de la misma manera que villano se realizara un override a algunos mensajes de supervillano como image(), colisionConBala(bala), moverse() y spawnearObjeto(), también supervillano tiene otras características que no tiene villano como lo son vivo y muerto

Texto

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Las propiedades y mensajes que se comportan igual son position,vida,danio,dirección,esAtrabesable,esGolpeable(),chocasteCarpincho(unPersonaje),hacerDanio() y moverseHacia(newDireccion)

Composicion

Un ejemplo de composición puede ser cuando interactúan los objetos de personaje y barraVida1 respectivamente. Originalmente se había creado la clase Personaje, en el que este puede tiene asignada una barra de vida en concreto como variable y cuando se produce el mensaje personaje.actualizarVida(), este manda el mensaje barraDeVida.recorrerLista(indice) que será recibida por su barra de vida. Es decir, son dos instancias distintas en la que una (personaje) conoce a la otra (barraDeVida1) y por ello de envía mensajes.

La razón por la que no se usó herencia en este caso fue por que no era necesario que la barra de vida conociera el comportamiento del personaje, y por qué al definirlo con composición, permitimos que en un futuro la barra de vida herede otra clase con la que tenga más en común (en este caso, la clase BarraDeVida)

