Documento: Buenas prácticas y conceptos del paradigma orientado a objetos aplicados al trabajo práctico integrador

Polimorfismo: Uno de los mensajes polimórficos del juego es, el método *recibirDisparo(unaBala)*. Éste método lo entiende la clase “BloqueDeHierro”, la sub-clase *“Enemigo”, “Columna”*, y los objetos *“personaje”* , *“stalin”, “zonaDeRescate”*, “avisoDeRescate”, “puerta” y “palanca”; que aunque sea el mismo mensaje, cada uno lo implementa de diferente manera. Otro método polimórfico es *chocarConStalin()*, ya que la sub-clase *“Enemigo”* , el objeto *“Personaje”*, *“zonaDeRescate”*, “avisoDeRescate” y “palanca”entienden el mensaje y lo implementan de maneras distintas.

Colecciones: Una de las colecciones utilizadas, es la ***listaDeEnemigos***, en el archivo *“saveStalin.wlk”*, en la que se instancian a los enemigos y se les asigna sus atributos a cada uno, luego en el método *agregarPersonajes()*, a esta lista se le envía un mensaje *forEach()*, en el que cada enemigo es mostrado en pantalla mediante un mensaje *addVIsual()*. Posteriormente en el método *configurarAcciones()*, mediante un mensaje *onTick()* al objeto *game*, se establece que cada 7000 milisegundos, cada enemigo de la lista recibe el mensaje *disparar()*, mediante un mensaje *forEach()* enviado a la *listaDeEnemigos*, de esa forma cada enemigo dispara en la dirección en la que está mirando (atributo especificado en la instanciación). Luego en el método *actualizarEnemigosVivos(enemigoMuerto)*, se extrae de la *listaDeEnemigos* a aquella instancia de *Enemigo* que ya no figure más en el juego.

De la misma forma, se utiliza una lista ***armasEnElPiso***, en la que se instancian armas que posteriormente se agregan al juego con el método *dropearArmas()*, de forma análoga a cuando se agregó la imagen de cada enemigo instanciado.

Clases: Utilizamos clases para no tener que codificar todos los objetos manualmente, y así evitar repetir lógica. Las clases son como moldes, que se asignan creando variables o constantes que instancien los objetos que parten de las mismas. Estás se crean en tiempo de ejecución. Un ejemplo de instanciación es en el archivo *“saveStalin.wlk”*, instanciamos todos los enemigos en una lista (línea 10), y luego, en la línea 52, a cada enemigo le agregamos su respectivo *add.Visual()*. Con esto logramos que aparezcan muchos enemigos en el juego, en tiempo de ejecución.

Herencia: Un ejemplo de utilización de herencia es, en el archivo *“armas.wlk”*, el objeto ***“Pistola”*** que hereda los atributos y los métodos de la clase ***“Arma”***; al igual que la clase ***“Ak47”*** que hereda de la clase ***“Arma”*** también. Evaluando el caso de Herencia entre el objeto ***“Pistola”*** y la clase ***“Arma”***, el comportamiento común entre ambos es todos los atributos de la clase y todos los métodos excepto

restarMunición(). En el caso de Herencia entre la clase ***"Ak47"*** y la clase ***"Arma"***, tienen todo en común.

Composición: Se utiliza para definir el movimiento del objeto *Stalin*, la dirección y posición de este objeto se le delegan a la variable *movimiento*, esta variable referencia inicialmente al objeto *prisionero*, en el que los métodos *position()* y *direccion()* retornan valores fijos. Luego, una vez que la variable referencia al objeto *escapar*, sus métodos *position()* y *direccion()* van a depender de la dirección y posición del objeto *Personaje*. El concepto de composición es más adecuado al de herencia porque si se hiciera por herencia, habría que instanciar a *Stalin* 2 veces, la primera vez cuando está con posición y dirección fijos, la segunda vez cuando sigue al *Personaje*, y la única diferencia que habría entre ellos, sería las mencionadas posición y dirección, es decir, no es flexible a cambios, por lo que se elimina uno y se instancia uno nuevo, mientras que por composición se delegan esas diferencias a los objetos *Prisionero* y *Escapar*, manteniendo al mismo objeto *Stalin*, sin instanciaciones innecesarias.