Documentación sobre las buenas prácticas y conceptos aplicados

Polimorfismo:

El polimorfismo nos permitió que objetos o clases distintas entiendan el mismo mensaje pero actúen de forma distinta. Un claro ejemplo de polimorfismo en el código es el method chocarJugador(), el cual lo implementan todos los objetos que se crean a partir de las clases Enemigos (tanto verticales como horizontales), los Choris (recargan la energía), los aviones (que en caso de que el pasaporte haya sido encontrado le permite al jugador pasar de nivel) y también el pasaporte. El objeto que se aprovecha de ello es el jugador ya que nos permite producir todos los efectos sobre el mismo. La ventaja del polimorfismo del mensaje chocarJugador() es que es gracias a este mensaje es que todos los objetos del juego producen un efecto sobre el jugador o el juego al ser colisionados por el jugador, y también en caso de querer agregar otro objeto visual que tenga un efecto al colisionar con el jugador es cuestión de simplemente agregar un objeto o clase nueva que entienda este method.

Colecciones:

Con respecto a las operaciones de colecciones, estas fueron utilizadas principalmente en la generación de los muros.Se hace uso del mensaje forEach en el método generarMuros() donde a partir de una lista de números que definimos por compresión (por ejemplo: [2..7]) generamos tanto líneas verticales como horizontales de muros. También se utilizó el mensaje any, en este caso sin efecto, en el mensaje jugador.puedeMoverse(unaOrientacion) donde evaluamos si los objetos en una cierta dirección son atravesables o no, fundamental para evitar que el jugador pueda pasar por encima de los muros.

Clases:

El uso de clases fue muy útil en el desarrollo del proyecto ya que fueron utilizadas en la gran mayoría de los distintos elementos implementados.Su uso se puede ver en las clases de los enemigos, los choris que dan energía al jugador, las puertas y el pasaporte. El motivo de esto es que al hacer un nivel era mucho más sencillo y cómodo crear en cada una de estas instancias cada una de las clases con la posición que nosotros queramos en vez de tener que hacer un objeto por cada enemigo en el juego. La implementación de clases nos permite además, evitar la repetición de lógica en el código ya que para cada enemigo nuevo que se crea va a tener todos los métodos que tenga la clase pero con valores distintos.

Herencia:

La implementación de herencia fue utilizada mayormente en los niveles y en los enemigos, principalmente en estos últimos debido a que los enemigos tanto verticales como

horizontales tienen como única diferencia a lo que devuelve el método moverse().Los demas parametros y metodos son comunes para ambos por lo que haciendo una superclase Enemigos y haciendo que los dos tipos hereden de ella logramos que tengan en común :

-Las variables de posición y límite
-Los métodos de chocarJugador() y esAtravesable().

De esta forma logramos evitar la repetición de lógica en cada una de las clases de enemigos verticales y horizontales.
En cuanto a los niveles, creamos una superclase Nivel con el método iniciar donde se cargan las visuales de la barra de vida y energía que son comunes para todos los niveles y luego en cada uno de ellos dentro del método iniciar() hacíamos un super() lo que nos permitió evitar tener que agregar cada una de esas visuales por separado en cada uno de los distintos niveles.

Composición:

El uso de composición fue utilizado en dos instancias distintas: en aviones y en el jugador.

En los aviones usamos composición para que cada avión sepa cual es el nivel que le sigue y cuando el jugador cumpla los requisitos para pasar de nivel, este lo inicie.De esta forma el avión le delega la tarea de iniciarse al nivel.

Con el jugador pasa algo similar.El jugador guarda el nivel en el que se encuentra actualmente y cada vez que colisiona con un enemigo, este pierde una vida y vuelve a iniciar el nivel y de esta forma “resetear” todo para poder volver a intentarlo.

Para estos casos no se utilizó herencia porque su uso resultaba imposible dado a que los objetos que se compusieron utilizan mensajes y variables que las otras clases u objetos no usan o no entienden y que tampoco tendría sentido que sepan ya que no las utilizarían.