× Se tuvo que agregar la librería de sonido aparte ya que no estaba incluida dentro del repositorio.(ya esta en el repositorio)

× La cantidad de vida inicial es excesiva. Probablemente esto haya quedado mal configurado luego

de efectuar la etapa de testeo.

× Tener métodos exclusivos para cierta funcionalidad no es una buena forma de implementar. Es

el caso de “congelaTiempo()”. Se deben usar patrones de diseño para poder lograr el efecto.(Se uso un objeto Phantom que es un SIngleton que se encarga de congelar enemigos, que lo crea Unicamente el powerUP )

× Lo mismo con mejorarDisparo. Se deben proveer los setters y getters necesario para evitar tener

que generar un método exclusivo. Por ejemplo, getArma podría devolverla y luego que el power

up sea el responsable de cambiarle la munición mejorándola, o en su defecto cambiarle el

estado, o cambiar el arma por otra.

× No queda claro el sentido de la clase powerUpRun.(hilo que mueva powerUP, que se duerma cuando este inactivo)

× En UML se utiliza itálica para representar las clases abstractas. Los \* son una convención de la

materia que facilitan la escritura a mano.(Acomodar el diagrama)

× El método Mapa2 figura en el diagrama de la clase Juego pero no en la implementación.(eliminar del diagrama el método Mapa2 de juego)

× Los métodos “habilitado” y “puedo” no son descriptivos y no se entiende a simple vista su

funcionalidad.

× El uso del casteo en la clase visitorDisparoP es innecesario. En la materia deben tomarse las

precauciones necesarias para evitar estos mecanismos.

× El paquete “misc” no debería existir. Cada clase contenida en dicho paquete debe estar en otro

junto con sus implementaciones concretas, o en su defecto, sus descendientes(creo paquete visitor y restructuo paquete nave)

× No está claro el sentido del atributo DISPARAR en la clase Uresunidad.(Era en nave, ya esta solucionado)

× Línea 105 clase jugador: moviendo se setea a false dos veces.(Luego de disparar se setea, es para evitar que se dispare de forma ininterrumpida)

× No queda claro por qué ele visitor retorna un booleano.(Es usado para saber si se puede o no mover a la celda)

× visitPlayer de visitorPowerUp: se supone que primero se debe accionar y luego destruir. Lo

mismo en visitDisparoPlayer.(corregido)

× En el visitor a determinados métodos se los implementa por ambos lados, aunque no se los

llame. Por ejemplo, visitEnemigo de la clase visitorPlayer y visitPlayer de la clase visitorEnemigo.

Ambos reducen la vida en 50 aunque solo se invoque visitPlayer.(preguntar a fede)

× Se deben evitar el “hardcoding”. Por ejemplo, en el enemigo Temporal se utiliza el valor 130 al

cambiar la estrategia.(Se creo una Constante en el Temporal Llamada VidaMax , cuando la vida actual es menor al 50% de vidaMax se hace el cambio de inteligencia).

× En los Visitor se repite todo el tiempo “j.getIsRunning()&&objeto.getIsRunning()”. Existen

muchas formas de evitar repetir código, una de ellas es invocando a super.(hay un método general ahora llamado areRunning(Gob o) en visitor, encargado de hacer ese checkeo)