HUNDIR LA FLOTA:
Feisty Fishermen

INTRODUCCIÓN:

En este proyecto se nos ha pedido diseñar e implementar el juego de “Hundir la flota”, cumpliendo una serie de requisitos y utilizando Java8 y protocolos donde sea posible. Con este fin hemos trabajado en grupos. El trabajo de cada miembro del grupo será detallado en los siguientes apartados de este documento.

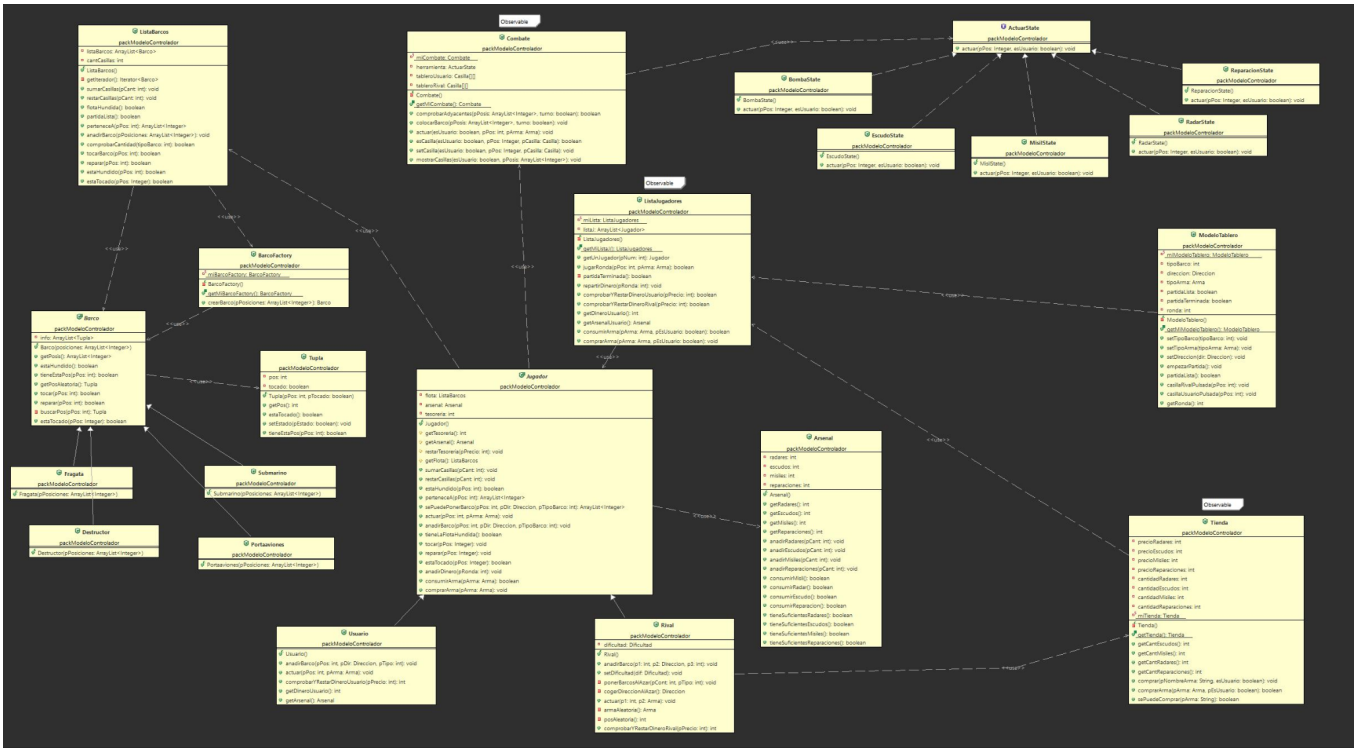
GESTIÓN:

A lo largo del proyecto los integrantes de nuestro grupo no han tenido un rol definido como tal, sino que en su lugar al comienzo de cada sprint separamos todas las tareas que había que completar y se asignaban de manera más o menos equilibrada entre los miembros del grupo, qué tareas han acabado asignadas y a quién se pueden ver más adelante en el apartado de desarrollo donde adicionalmente indicamos cuánto tiempo ha sido dedicado a estas.
En lo que respecta a la comunicación entre nosotros, no han sido necesarias más reuniones que las correspondientes a las horas de laboratorio, ya que mediante la utilización de WhatsApp siempre podíamos contactar al grupo para resolver dudas o conflictos y si algún problema requería nuestra presencia física lo podíamos atender a la salida de clases porque todo el grupo pertenecemos al mismo subgrupo de laboratorios.
Finalmente, más allá de los apuntes que se pueden encontrar en eGela y la ocasional visita a Stack Overflow no ha sido necesaria la utilización de recursos adicionales.

DISEÑO:

DIAGRAMA DE CLASES:

Sin enums:



Con enums:

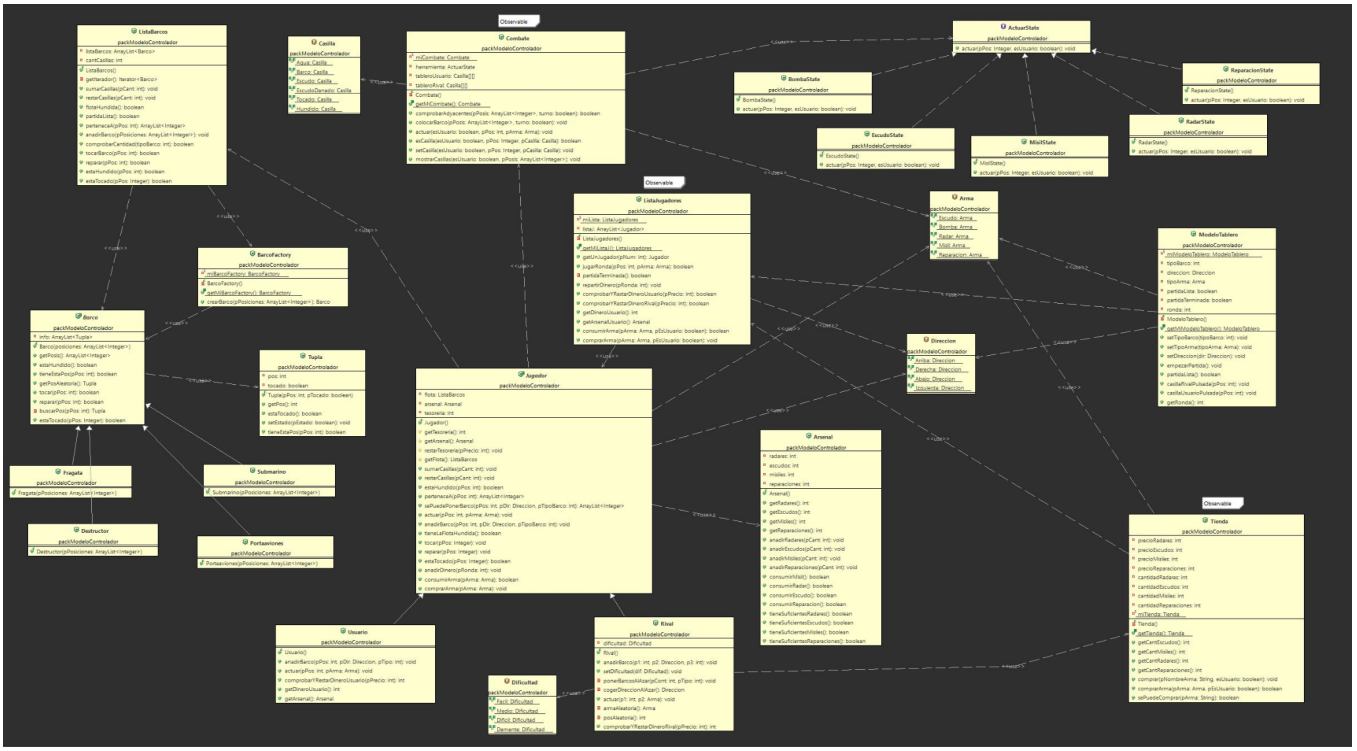
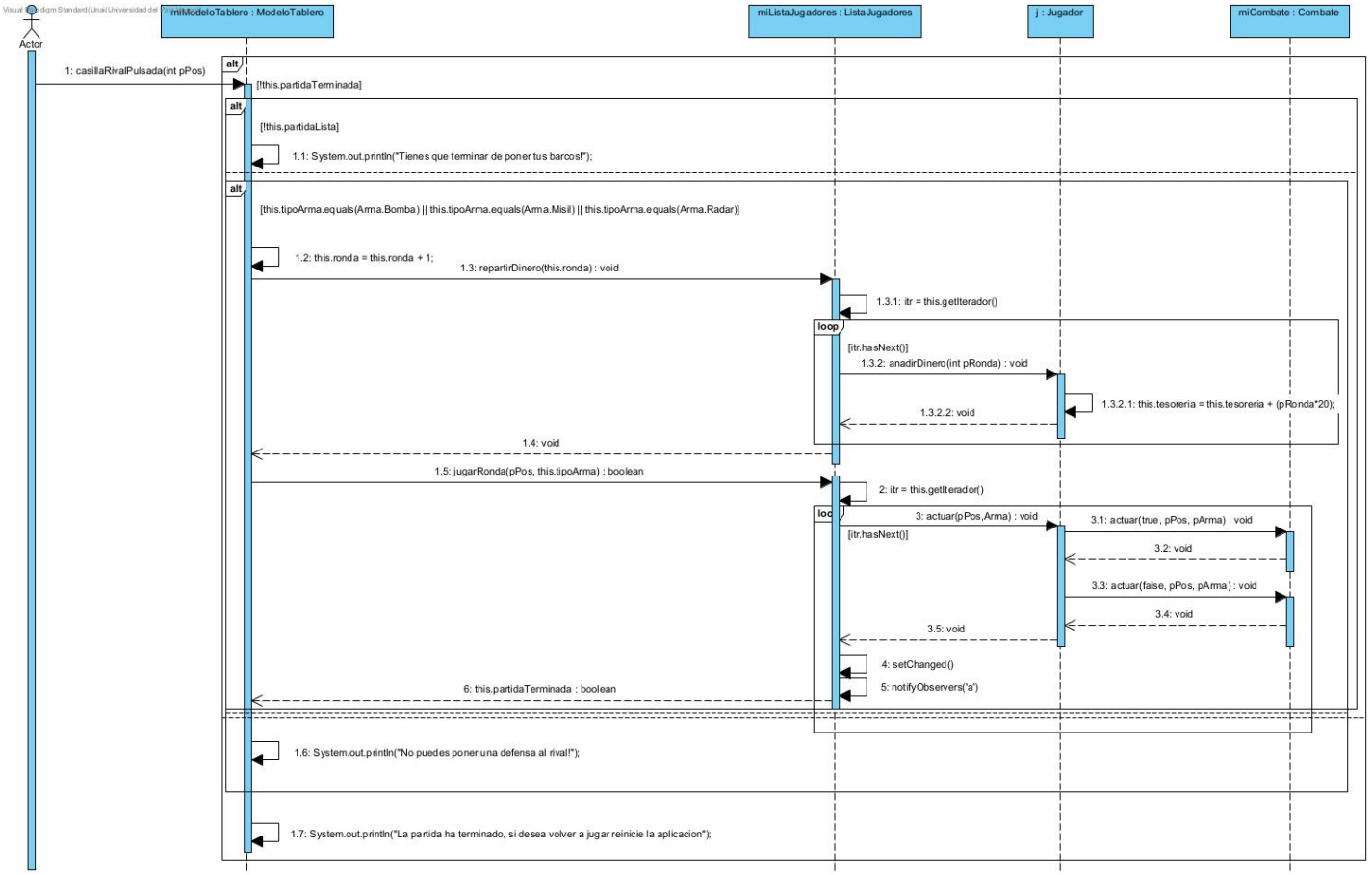
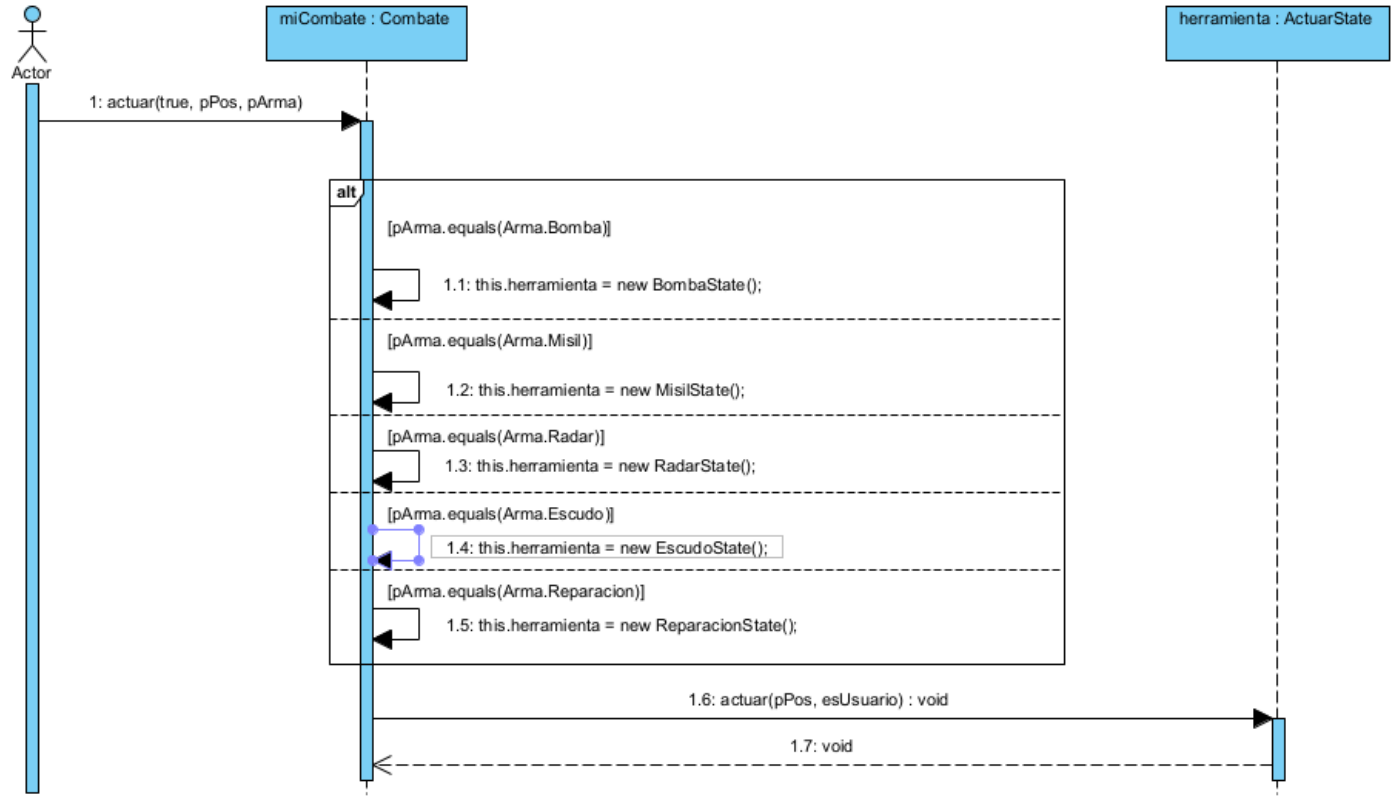


DIAGRAMA DE SECUENCIA (ATACAR USUARIO):

Los diagramas se hicieron antes de cambiar el nombre de algunas clases, todos los Strategy son los State que aparecen en el diagrama de clases.





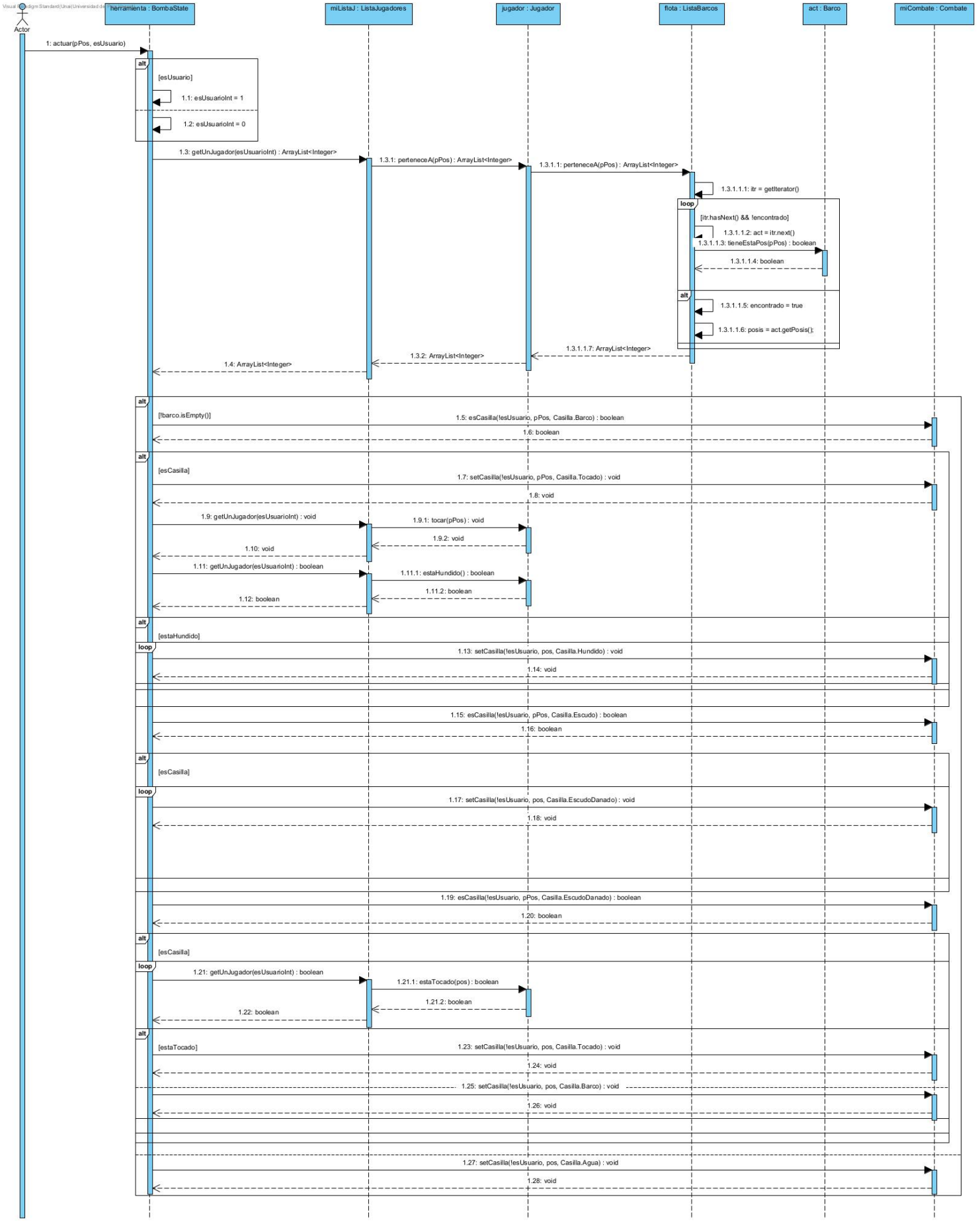
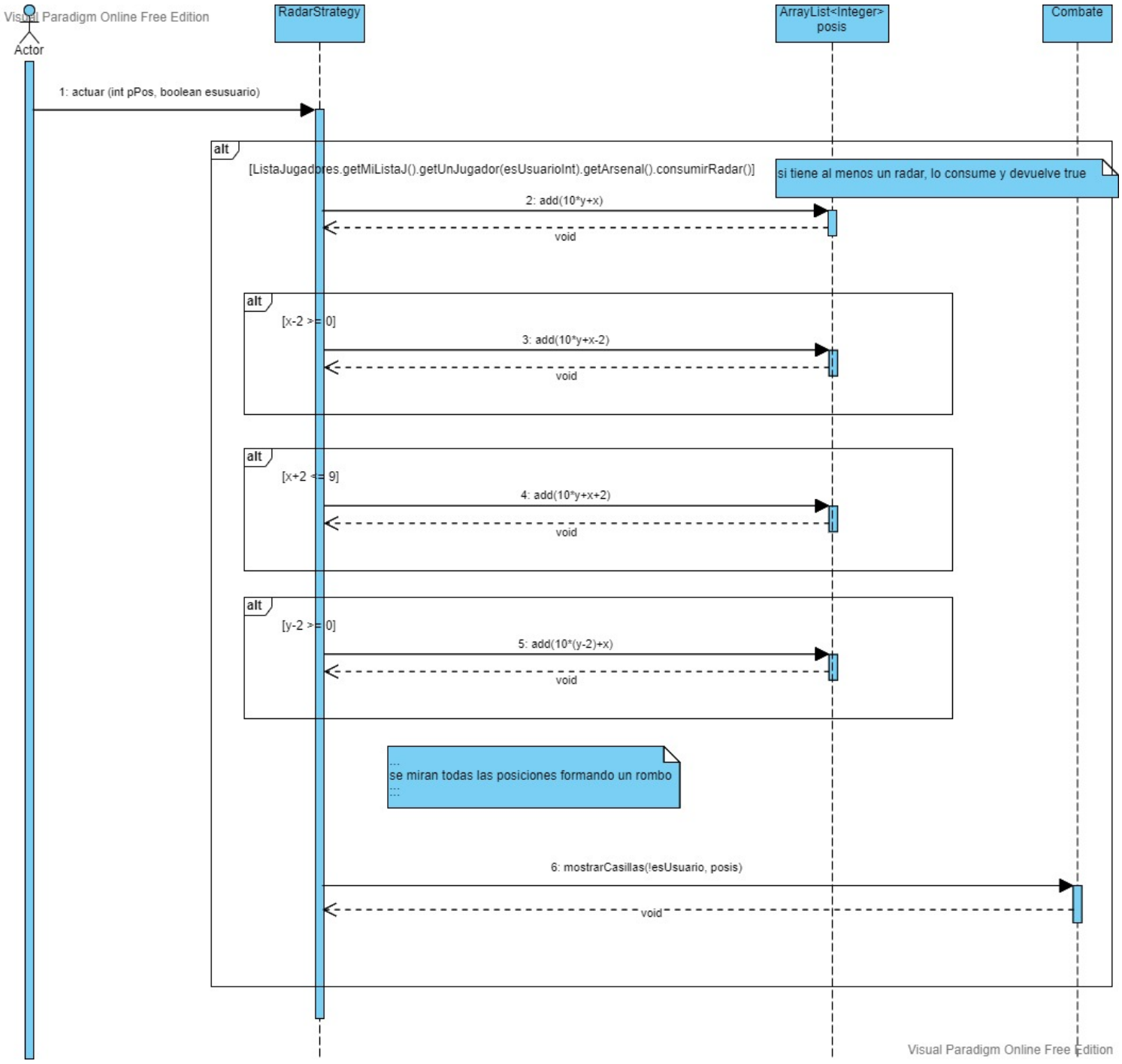

Actor
herramienta : BombaState
miListaJ : ListaJugadores
jugador : Jugador
flota : ListaBarcos
act : Barco
miCombate : Combate
1: actuar(pPos, esUsuario)
alt
[esUsuario]
1.1: esUsuarioInt = 1
1.2: esUsuarioInt = 0
1.3: getUnJugador(esUsuarioInt) : ArrayList<Integer>
1.3.1: perteneceA(pPos) : ArrayList<Integer>
1.3.1.1: perteneceA(pPos) : ArrayList<Integer>
1.3.1.1.1: itr = getIterator()
loop
[itr.hasNext() && !encontrado]
1.3.1.1.2: act = itr.next()
1.3.1.1.3: tieneEstaPos(pPos) : boolean
1.3.1.1.4: boolean
alt
1.3.1.1.5: encontrado = true
1.3.1.1.6: posis = act.getPosis();
1.3.1.1.7: ArrayList<Integer>
1.3.2: ArrayList<Integer>
1.4: ArrayList<Integer>
alt
[!barco.isEmpty()]
1.5: esCasilla(!esUsuario, pPos, Casilla.Barco) : boolean
1.6: boolean
alt
[esCasilla]
1.7: setCasilla(!esUsuario, pPos, Casilla.Tocado) : void
1.8: void
1.9: getUnJugador(esUsuarioInt) : void
1.9.1: tocar(pPos) : void
1.9.2: void
1.10: void
1.11: getUnJugador(esUsuarioInt) : boolean
1.11.1: estaHundido() : boolean
1.11.2: boolean
1.12: boolean
alt
[estaHundido]
loop
1.13: setCasilla(!esUsuario, pos, Casilla.Hundido) : void
1.14: void
1.15: esCasilla(!esUsuario, pPos, Casilla.Escudo) : boolean
1.16: boolean
alt
[esCasilla]
loop
1.17: setCasilla(!esUsuario, pos, Casilla.EscudoDanado) : void
1.18: void
1.19: esCasilla(!esUsuario, pPos, Casilla.EscudoDanado) : boolean
1.20: boolean
alt
[esCasilla]
loop
1.21: getUnJugador(esUsuarioInt) : boolean
1.21.1: estaTocado(pos) : boolean
1.21.2: boolean
1.22: boolean
alt
[estaTocado]
1.23: setCasilla(!esUsuario, pos, Casilla.Tocado) : void
1.24: void
1.25: setCasilla(!esUsuario, pos, Casilla.Barco) : void
1.26: void
1.27: setCasilla(!esUsuario, pos, Casilla.Agua) : void
1.28: void

DIAGRAMA DE SECUENCIA (RADAR RIVAL):



DESARROLLO:

SPRINT 1:

El primer sprint es en el que más tiempo teníamos principalmente porque también era necesario implementar toda la interfaz gráfica. Los principales objetivos de este sprint fueron conseguir que “Hundir la flota” fuera jugable, cumpliendo con las peticiones para este sprint que simplemente eran que se pudiera atacar y poner barcos. Adicionalmente para esta versión pudimos implementar una manera de ajustar la dificultad de la partida, haciendo que la I.A. pudiera saber en ocasiones donde se encontraban los barcos del usuario. Las tareas que se asignaron fueron las siguientes:

Diseño e implementación de BaseTablero y PopUpGanador:
Encargado: Rubén Esgueva
Tiempo estimado: 2h **Tiempo requerido:** 1h 30 mins
Comentarios:
Como este apartado no incluye el controlador ni el update era básicamente añadir las cosas en el Display, lo cual no llevaba mucho tiempo.

Diseño e implementación de MainMenu:
Encargado: Unai Elorriaga
Tiempo estimado: 1h **Tiempo requerido:** 1h 30 mins
Comentarios:
La única función de esta vista es permitir la selección de dificultad, tiene su propio controlador y este sí que se ha implementado en este apartado por lo que ha llevado un poco más de tiempo que el anterior.

Implementación del controlador de BaseTablero:
Encargado: Unai Elorriaga y Rubén Esgueva
Tiempo estimado: 1h **Tiempo requerido:** 45 mins
Comentarios:
Una vez sabes como funciona el patrón MVC la implementación del controlador es sencilla.

Update de la vista:
Encargado: Unai Elorriaga y Rubén Esgueva
Tiempo estimado: 30 mins **Tiempo requerido:** 45 mins
Comentarios:
Pasó por un par de versiones antes de encontrar una que nos convenciera, pero ha sido relativamente simple de implementar y no nos ha llevado mucho tiempo.

Implementación de las clases Barco y Tupla:
Encargado: Jon Robledo
Tiempo estimado: 1h **Tiempo requerido:** 1h 30 mins
Comentarios:
Se trata de dos clases simples con métodos no muy largos y que no tienen más conexiones que entre sí. La mayor parte del tiempo se corresponde a modificaciones para que otras clases funcionen correctamente.

Algoritmo para poner barcos del Usuario:
Encargado: Unai Elorriaga, Rubén Esgueva y Eder Sampayo
Tiempo estimado: 1h 30 mins **Tiempo requerido:** 2h 45 mins
Comentarios:
Hacer funcionar este algoritmo ha sido más complicado de lo que a simple vista parecía, ya que no solo hay que colocar los barcos sino que previamente hay que detectar las posiciones imposibles y comprobar la adyacencia.

Algoritmo para poner barcos del Rival:
Encargado: Unai Elorriaga
Tiempo estimado: 1h **Tiempo requerido:** 45 mins
Comentarios:
En este apartado simplemente se adaptaron algunos de los métodos utilizados por el usuario para colocar sus barcos, por lo que no era tan intensivo en lo que a código se refiere.

Algoritmo para comprobar si hay algún ganador:
Encargado: Rubén Esgueva y Eder Sampayo
Tiempo estimado: 30 mins **Tiempo requerido:** 20 mins
Comentarios:
Como solo hay que mirar los barcos restantes de cada jugador, no se necesita mucho código y ha sido sencillo de implementar.
Algoritmo para atacar (Usuario e I.A.):
Encargado: Todo el grupo
Tiempo estimado: 1h 30 mins **Tiempo requerido:** 1h 30 mins
Comentarios:
Si bien el algoritmo empleado en esta versión no es muy complicado, hay muchas cosas a tener en cuenta y aunque no sea un algoritmo muy largo todos los integrantes del grupo han ayudado a su implementación de una manera u otra.

Informe MVC:
Encargado: Rubén Esgueva
Tiempo estimado: 30 mins **Tiempo requerido:** 30 mins

Informe de tareas:
Encargado: Rubén Esgueva
Tiempo estimado: 1h **Tiempo requerido:** 1h 30 mins

Diagrama de clases:
Encargado: Eder Sampayo
Tiempo estimado: 1h **Tiempo requerido:** 45 mins

Diagrama de secuencia:
Encargado: Jon Robledo
Tiempo estimado: 1h **Tiempo requerido:** 1h

SPRINT 2:

En el segundo sprint el diseño del primero fue sometido a grandes cambios, creamos multitud de clases nuevas con el fin de modularizar el programa y esto se refleja en el tiempo dedicado a cada tarea, ya que la mayor parte de este SPRINT ha sido dedicada a la implementación de las nuevas clases. Adicionalmente se aprovechó este SPRINT para arreglar algunos bugs que se encontraron al presentar el primero y se cumplieron también con las demandas de este, las cuales eran la implementación de las herramientas radar y escudo. Las tareas que se asignaron fueron las siguientes:

Uso de escudos para el Usuario y para el Rival:
Encargado: Unai Elorriaga y Rubén Esgueva
Tiempo estimado: 30 mins **Tiempo requerido:** 2h aprox.
Comentarios:
Hemos tardado bastante más en implementar los métodos asociados a esta función porque inicialmente malinterpretamos el objetivo, pensando que un escudo solo cubría una casilla en lugar de todo el barco. Por lo que acabamos teniendo que modificar y añadir otros varios métodos para poder saber a qué barco estaba asociada la casilla en la que se ponía el escudo. Además hubo que hacer múltiples intentos para que si se pone un escudo en un barco dañado no solo se permita sino que se recuerde qué casillas de ese barco estaban tocadas previamente.

Uso de radar para el Usuario y para el Rival:
Encargado: Rubén Esgueva
Tiempo estimado: 30 mins **Tiempo requerido:** 1h aprox
Comentarios:
Inicialmente esta función parecía muy simple, pero debido a cómo teníamos codificadas las diferentes armas, hubo que reestructurar el método que usamos para llamarlas, ya que de lo contrario el radar solo funcionaría si al hacer click en el tablero se acierta en una casilla de tipo barco.

Descentralización de ModeloTablero:
Encargados: Todo el grupo
Tiempo estimado: 5h aprox. (cada miembro) **Tiempo requerido:** 8h 40 mins aprox.
Comentarios:
Esta sección ha sido el trabajo principal a lo largo de este sprint. En este primer apartado hemos indicado el tiempo en general pero ahora a continuación profundizaremos más en los subapartados que lo forman.

Actualización de Observable y Observer:
Encargado: Rubén Esgueva
Tiempo estimado: 30 mins **Tiempo requerido:** 20 mins
Comentarios:
Como hemos generado una gran cantidad de clases y hemos repartido los métodos entre ellas, qué clases se consideraban Observables ha cambiado, por lo que había que actualizar los updates y los notifyObservers() de acuerdo con la nueva configuración. Si bien el método Update() en el Observer puede resultar intimidante de ver, en realidad no es muy complejo por lo que no fué necesario mucho tiempo para tenerlo funcionando.

Extraer MainMenu de BaseTablero:
Encargados: Unai Elorriaga y Jon Robledo
Tiempo estimado: 30 min **Tiempo requerido:** 30 mins
Comentarios:
Como MainMenu era una clase privada dentro de BaseTablero, solo ha tenido que extirparse y cambiar de sitio el método main() para que funcione al tener las dos clases separadas.

Creación de ListaJugadores, Jugador, Usuario y Rival:
Encargados: Rubén Esgueva, Jon Robledo y Eder Sampayo
Tiempo estimado: 1h (cada miembro) **Tiempo requerido:** 2h 30 mins
Comentarios:
Este conjunto de clases lleva la cuenta del estado de cada jugador de la partida. Los métodos de poner barcos y atacar/defender han sido alterados en gran medida respecto al primer sprint por lo que no se podía reutilizar bastante parte del código. Además de esto, la clase Combate requiere información proporcionada por estas por lo que en múltiples ocasiones han tenido que ser modificadas para añadir métodos en cascada.

Limpiar ModeloTablero:
Encargado: Eder Sampayo
Tiempo estimado: 15 mins **Tiempo requerido:** 10 mins
Comentarios:
Al mover la mayoría de los métodos a otras clases, muchas funciones quedaban inutilizadas, por lo que por claridad había que borrar los métodos que ya no servían.

Añadir herencia a los Barcos:
Encargado: Eder Sampayo
Tiempo estimado: 30 mins **Tiempo requerido:** 10 mins
Comentarios:
La herencia no cumple un gran papel en la estructura de nuestro programa, por lo que las clases que heredan de Barco no tienen más que una constructora, esto hizo que la creación de las distintas subclases no llevara mucho tiempo y este se invirtiera en su mayoría en modificar los métodos que hacían referencia al tamaño del barco para que en su lugar lo hagan al tipo de barco.

Creación y puesta a punto de Combate:
Encargados: Unai Elorriaga y Jon Robledo
Tiempo estimado: 2h (cada miembro) **Tiempo requerido:** 5h aprox.
Comentarios:
Combate tiene la importante función de llevar la situación de ambos tableros así como de actualizarlos y avisar a la vista de manera adecuada cada vez que se usa una arma/herramienta en alguna de las casillas. Los métodos comprobarAdyacentes(), atacar() y defender() han pasado por una gran cantidad de modificaciones y arreglos, haciendo que esta clase y los métodos de otras clases que necesita para funcionar hayan causado un gran consumo de tiempo.

Informe de tareas:
Encargado: Rubén Esgueva
Tiempo estimado: 1h **Tiempo requerido:** 1h 30 mins

Diagrama de clases:
Encargado: Eder Sampayo
Tiempo estimado: 1h **Tiempo requerido:** 45 mins

Diagrama de secuencia:
Encargado: Jon Robledo
Tiempo estimado: 1h **Tiempo requerido:** 1h

SPRINT 3:

El tercer SPRINT es el más corto, ya que no teníamos que generar tanto código como en los dos primeros. En este nos limitamos a implementar la tienda para que tanto el usuario como la I.A. puedan comprar, a crear la herramienta de reparación y a modificar el algoritmo de usar armas para que funcione con el patrón State. Las tareas que se asignaron fueron las siguientes:

Implementación de la tienda para el usuario y el rival:
Encargado: Jon Robledo y Eder Sampayo
Tiempo estimado: 2h **Tiempo requerido:** 4h
Comentarios:
Si bien implementar esta sección no altera mucho el código escrito previamente, como se trata de una función completamente nueva ha habido que escribir todo el código desde cero

Crear ActuarState y reducir el desplazar el código de Combate:
Encargado: Unai Elorriaga, Rubén Esgueva y Eder Sampayo
Tiempo estimado: 1h 30 mins **Tiempo requerido:** 3h aprox.
Comentarios:
Hemos tenido que mover y adaptar mucho código para hacerlo funcionar, pero no ha dado mayores problemas.

Algoritmo para la herramienta Reparación:
Encargado: Unai Elorriaga, Rubén Esgueva y Eder Sampayo
Tiempo estimado: 1h **Tiempo requerido:** 1h
Comentarios:
Ha habido algunos casos de prueba que nos llevó un tiempo arreglar pero no hemos tenido especial dificultad con esta herramienta.

Documentación final:
Encargado: Rubén Esgueva
Tiempo estimado: 3h **Tiempo requerido:** 5h

Diagrama de clases:
Encargado: Eder Sampayo
Tiempo estimado: 1h **Tiempo requerido:** 45 mins

Diagrama de secuencia (Atacar Usuario):
Encargado: Unai Elorriaga
Tiempo estimado: 1h **Tiempo requerido:** 1h 30 mins

Diagrama de secuencia (Radar Enemigo):
Encargado: Jon Robledo
Tiempo estimado: 1h **Tiempo requerido:** 45 mins

CASOS DE PRUEBA:

Como nuestro programa ha cambiado mucho entre el primer y el segundo sprint, las pruebas que hicimos en el primero no eran aplicables al segundo, por lo que solo vamos a tener en cuenta los casos observados de ahí en adelante.

Prueba: Colocar un barco en el tablero al comienzo de la partida
Resultado esperado: Se coloca el barco si se pueden colocar barcos de ese tipo, no va a quedar ninguna casilla fuera de la matriz y no se superpone ni es adyacente a ningún otro barco.
Comentarios: El único problema que originalmente tuvimos con el algoritmo que se encarga de esto es que las posiciones del estilo [4],[9] y [5],[0] las consideraba adyacentes por lo que no dejaba poner barcos que sí eran posibles, en la versión final esto está solucionado.

Prueba: Colocar un barco después de comenzar la partida
Resultado esperado: Se ignora la acción.
Comentarios: --

Prueba: Utilizar un arma/herramienta antes de acabar de colocar los barcos
Resultado esperado: Se ignora la acción.
Comentarios: --

Prueba: Utilizar un arma/herramienta cuando no hay stock en el Arsenal
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: Perder el turno es un poco brutal, pero si queremos darle una segunda oportunidad al jugador para realizar otra acción habría que cambiar mucho código por lo que hemos considerado que no merece la pena.

Prueba: Comprar un arma/herramienta en la Tienda cuando no queda stock
Resultado esperado: Se ignora la acción.
Comentarios: --

Prueba: Comprar un arma/herramienta en la Tienda cuando no se tiene dinero suficiente
Resultado esperado: Se ignora la acción.
Comentarios: --

Prueba: Comprar un arma/herramienta en la Tienda
Resultado esperado: Se añade la herramienta al Arsenal de quien lo ha comprado, se reduce su dinero en la cantidad indicada y se reduce el stock de la Tienda.
Comentarios: Para facilitar la implementación y añadir un aspecto de estrategia a la tienda, se comparte el stock de esta entre usuario y rival.

Prueba: Tratar de utilizar un arma/herramienta en el campo que no le corresponde (bomba/misil/radar en tu campo, o escudo/reparación en el del rival).
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: --

Prueba: Se usa el turno para atacar con una Bomba a una casilla Agua del rival
Resultado esperado: Se revela en la interfaz que la casilla era de tipo agua y el rival toma una acción.
Comentarios: El rival toma al azar la decisión de qué arma y en qué casilla utilizar, siendo el resultado de esto lo mismo que las acciones del usuario pero en el tablero opuesto.

Prueba: Se usa el turno para atacar con una Bomba a una casilla Barco del rival
Resultado esperado: La casilla pasa a ser de tipo Tocado o si con esa casilla se termina el barco pasa a Hundido y se notifica a la interfaz, después el rival toma una acción.
Comentarios: --

Prueba: Se usa el turno para atacar con una Bomba a una casilla Escudo/EscudoDañado del rival
Resultado esperado: Si es una casilla escudo todas las casillas que forman el barco protegido pasan a ser EscudoDañado, y si ya estaba dañado vuelven a su estado previo a poner el escudo, después el rival toma una acción.
Comentarios: --

Prueba: Se usa el turno para atacar con una Bomba a una casilla Tocado/Hundido del rival
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: --

Prueba: Se usa el turno para atacar con un Misil a una casilla Barco/Tocado del rival
Resultado esperado: Todas las posiciones ocupadas por ese barco pasan a estar hundidas y por cada posición que no estuviera previamente tocada el rival pierde una casilla de su contador de casillas totales.
Comentarios: --

Prueba: Se usa el turno para atacar con un Misil a una casilla Agua/Hundido del rival
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: --

Prueba: Se usa el turno para atacar con un Misil a una casilla Escudo/EscudoDañado del rival
Resultado esperado: Se elimina por completo el escudo y las casillas correspondientes al barco pasan a tener los valores que tenían antes de protegerse.
Comentarios: --

Prueba: Se usa el turno para usar el Radar en cualquier casilla del rival
Resultado esperado: Se actualiza la interfaz para enseñarle al jugador el contenido de las casillas en un rombo desde la que se ha clickeado y luego el rival ejecuta su turno.
Comentarios: El rival también puede utilizar su radar pero como no está atento a la interfaz no le sirve de nada.

Prueba: Se usa el turno para usar el Escudo en una casilla Barco/Tocado del usuario
Resultado esperado: Todas las posiciones ocupadas por ese barco pasan a estar protegidas y luego el rival ejecuta su turno
Comentarios: --

Prueba: Se usa el turno para usar el Escudo en cualquier otra casilla del usuario.
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: --

Prueba: Se usa el turno para usar una Reparación en una casilla Barco/Tocado.
Resultado esperado: Todas las casillas que estaban Tocadas del Barco pasan a ser de tipo Barco, y la cantidad de casillas reparadas se vuelve a añadir a las totales del usuario, después el rival juega su turno.
Comentarios: --

Prueba: Se usa el turno para usar una Reparación en una casilla Escudo/EscudoDañado.
Resultado esperado: Aunque no se vea desde la interfaz las casillas del barco protegido pasan por el mismo proceso que en el caso anterior.
Comentarios: Este caso nos causó el tener que modificar una cantidad de métodos de otras clases.

Prueba: Se usa el turno para usar una Reparación en una casilla Agua/Hundido.
Resultado esperado: Se ignora la acción del usuario y pierde el turno.
Comentarios: --

CONCLUSIONES:

Este proyecto ha sido una muy buena experiencia para los miembros de nuestro grupo. Evaluando el funcionamiento de nuestro grupo, podemos decir que estamos muy contentos con nuestra forma de trabajar. No hemos tenido ningún tipo de discusión y los desacuerdos se han podido resolver en un tiempo razonable y sin dificultad. Adicionalmente, estamos satisfechos con la cantidad de trabajo realizada por cada individuo y creemos que las labores se han repartido de forma justa sin nadie trabajando de más o de menos.

En lo que a los contenidos del proyecto respecta nos ha servido para trabajar los distintos patrones y protocolos estudiados a lo largo del curso, además nos ha enseñado a modularizar los programas para que las futuras versiones de estos sean más fáciles de implementar.

Hablando ahora de nuestro producto final, creemos que hemos propuesto una solución correcta que no solo cumple con las especificaciones dadas sino que además utiliza Java8 donde es posible y tres patrones distintos: Factory, State y Observer.
El principal cambio que se podría aplicar en un futuro sobre nuestra solución es el volver a implementar el sistema de dificultades que habíamos programado en el primer SPRINT, ya que nos parece un concepto muy interesante pero tras los enormes cambios a los que fue sometido nuestro programa el código que teníamos era inutilizable y no hemos encontrado el momento de incluirlo.

Para facilitar las pruebas con nuestro proyecto, estas son las instrucciones para jugar:

1. Al iniciar el juego se debe elegir una dificultad y pulsar el botón que aparece en el centro.
2. Antes de poder usar cualquier otra función se deberán poner los barcos. Para esto hay que seleccionar el botón correspondiente a los barcos que se quieran poner y la dirección en la que se orientan (No se reinicia una vez pulsada por primera vez, pero no tiene valor predeterminado).
3. Para colocar un barco simplemente se clickea en la casilla en la que se desea colocar, si no ocurre nada es porque esa posición no es adecuada.
4. Una vez se hayan puesto los barcos empieza la partida.
5. Al jugar las herramientas si tienen un valor predeterminado (Bomba) y si se quiere utilizar alguna otra habrá que seleccionarlas cada vez, ya que tras atacar el tipo de arma se reinicia al predeterminado. Hay una cantidad limitada de cada herramienta y si se desea adquirir más habrá que comprarlas en la tienda.
6. Para abrir la tienda habrá que clickear el botón de la parte superior izquierda de la pantalla. Una vez hecho esto se verá la cantidad de dinero que se posee y los productos que se pueden comprar (cabe destacar que el stock de la tienda es genérico para la partida, por lo que si un jugador compra todos los escudos el otro no podrá adquirir ninguno). Simplemente hay que seleccionar la herramienta que se desea obtener y si el jugador posee la cantidad de dinero suficiente esta se añadirá directamente al arsenal.