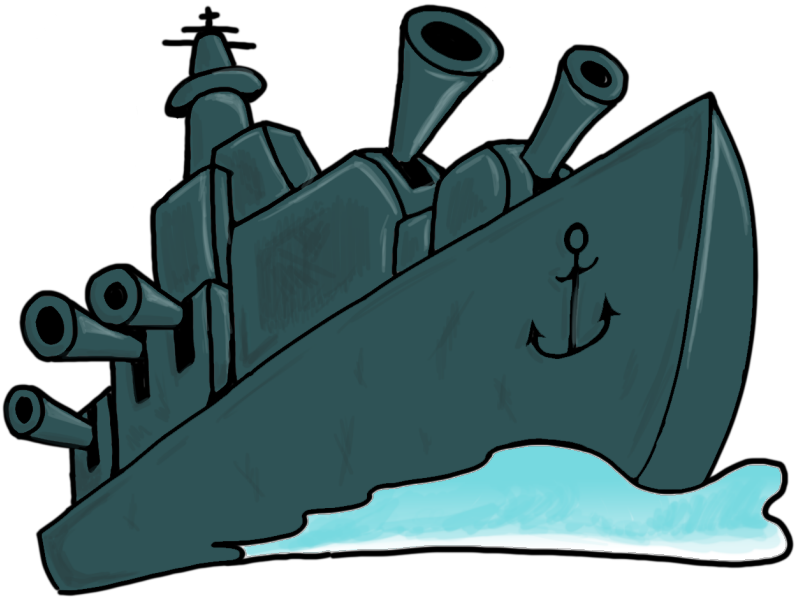


Juego de Batalla Naval

DISEÑO DEL SISTEMA

Grupo: La Osa que Baila

Integrantes: Martinez, Facundo Jesus

Mugni, Juan Mauricio

Reynoso Choque, Kevin Walter

2022

Historial de Cambios

Fecha	Resumen	Autor/es
5/6/2022	Primera presentación del informe realizado	Martinez, Facundo Jesus Mugni, Juan Mauricio Reynoso Choque, Kevin Walter
19/6/2022	Corrección de los puntos comentados y agregado de contenido faltante	Martinez, Facundo Jesus Mugni, Juan Mauricio Reynoso Choque, Kevin Walter

Índice

Diagrama de paquetes

A continuación se muestra la organización de los componentes del programa (tales como código y documentación).

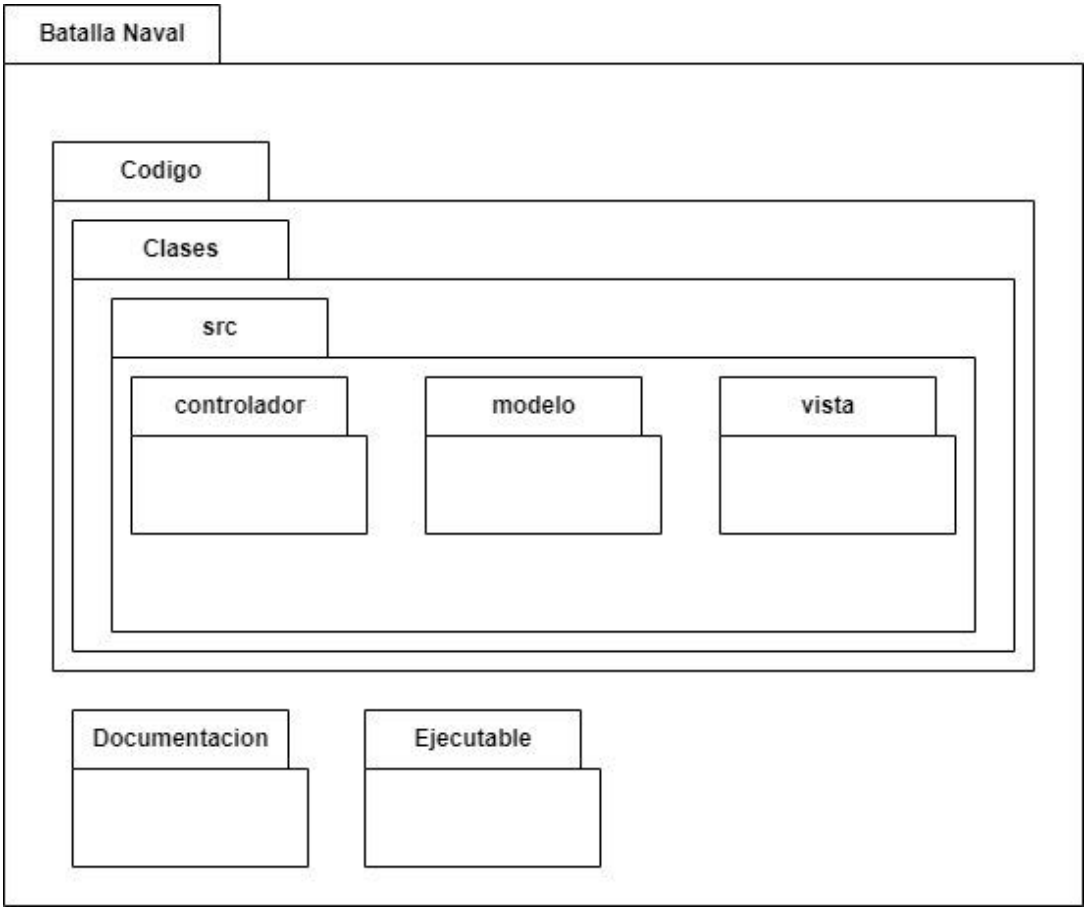


Diagrama de clases y objetos

Se muestran los distintos objetos que componen el juego y sus respectivas relaciones.

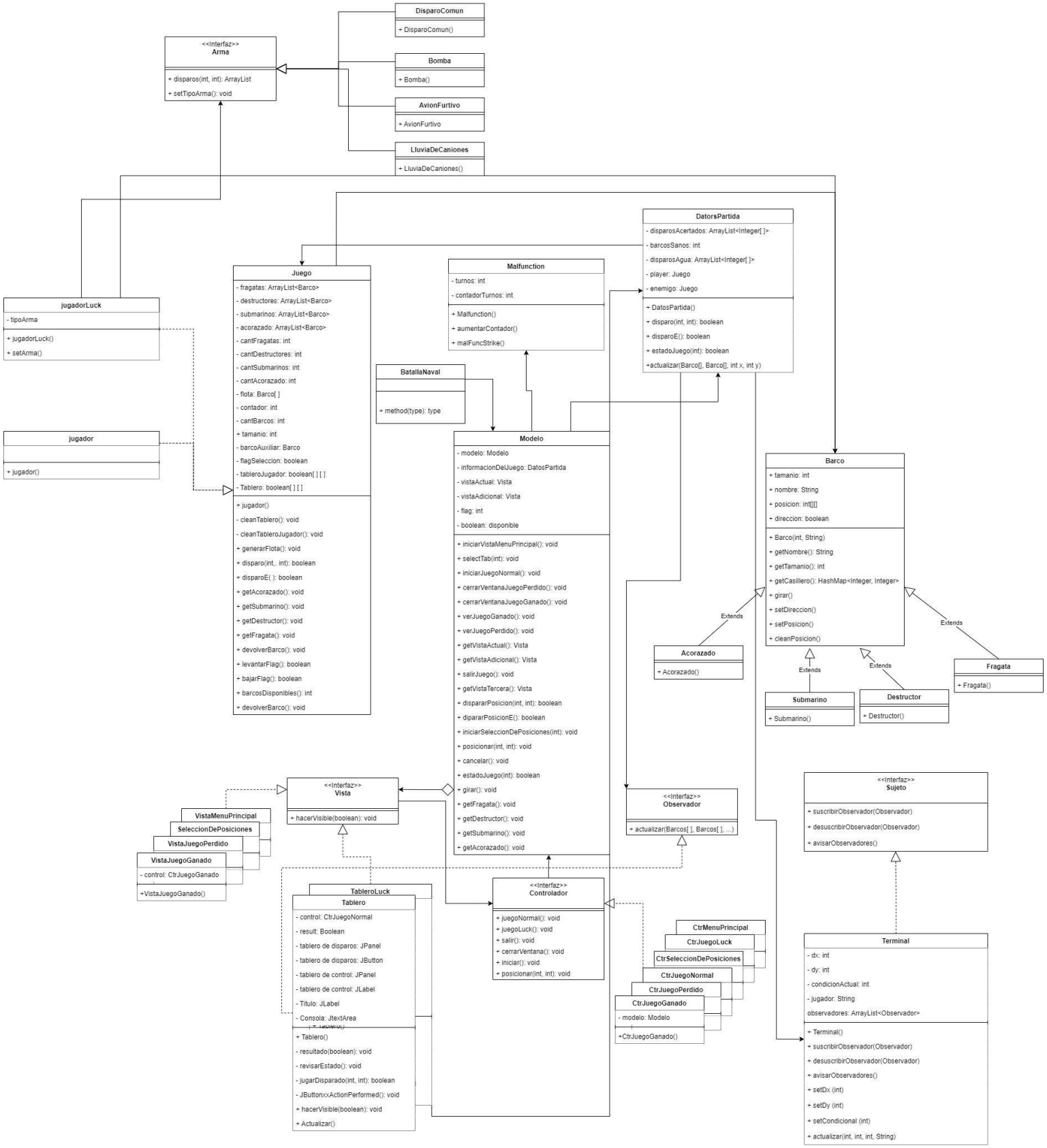
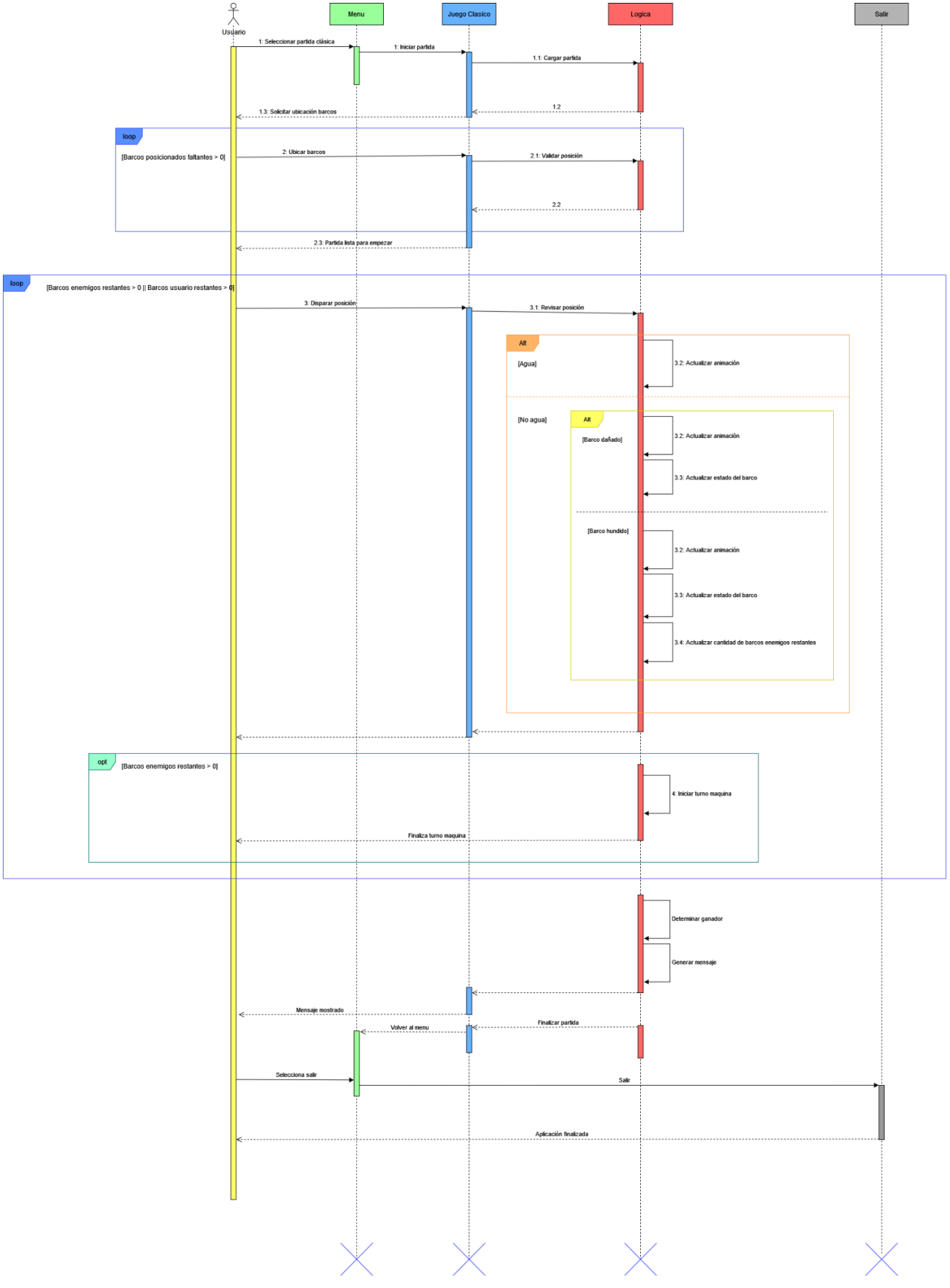


Diagrama de secuencia

El diagrama de secuencia realizado representa cuando el jugador inicia una partida, juega y sale.

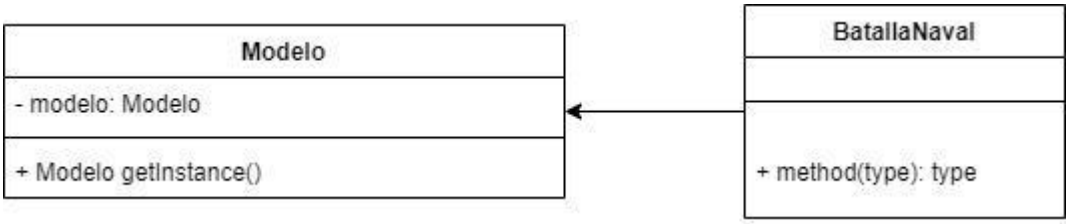
Jugador inicia una partida del modo clásico, juega y sale:

- El usuario selecciona el modo de juego Clásico/Normal
- El programa debe encargarse de cargar la correspondiente partida, abriendo la ventana de selección de posiciones
- Se le solicitará al usuario ubicar los barcos
- El usuario deberá ubicar todos los barcos disponibles y el programa debe validar las posiciones.
- Se le avisará al usuario que puede iniciar la partida.
- Actualizada la vista al tablero de juego de “modo clásico” el usuario deberá realizar disparos en su turno.
- El programa se encarga de determinar cuál fue el resultado del disparo.
- Una vez que se acaben los barcos del enemigo o del usuario, se debe enviar un mensaje con el resultado (“derrota” o “victoria”).
- Una vez enviado el mensaje, se finaliza la partida y, en este caso, el usuario vuelve al menú principal y selecciona la opción salir para finalizar el programa.

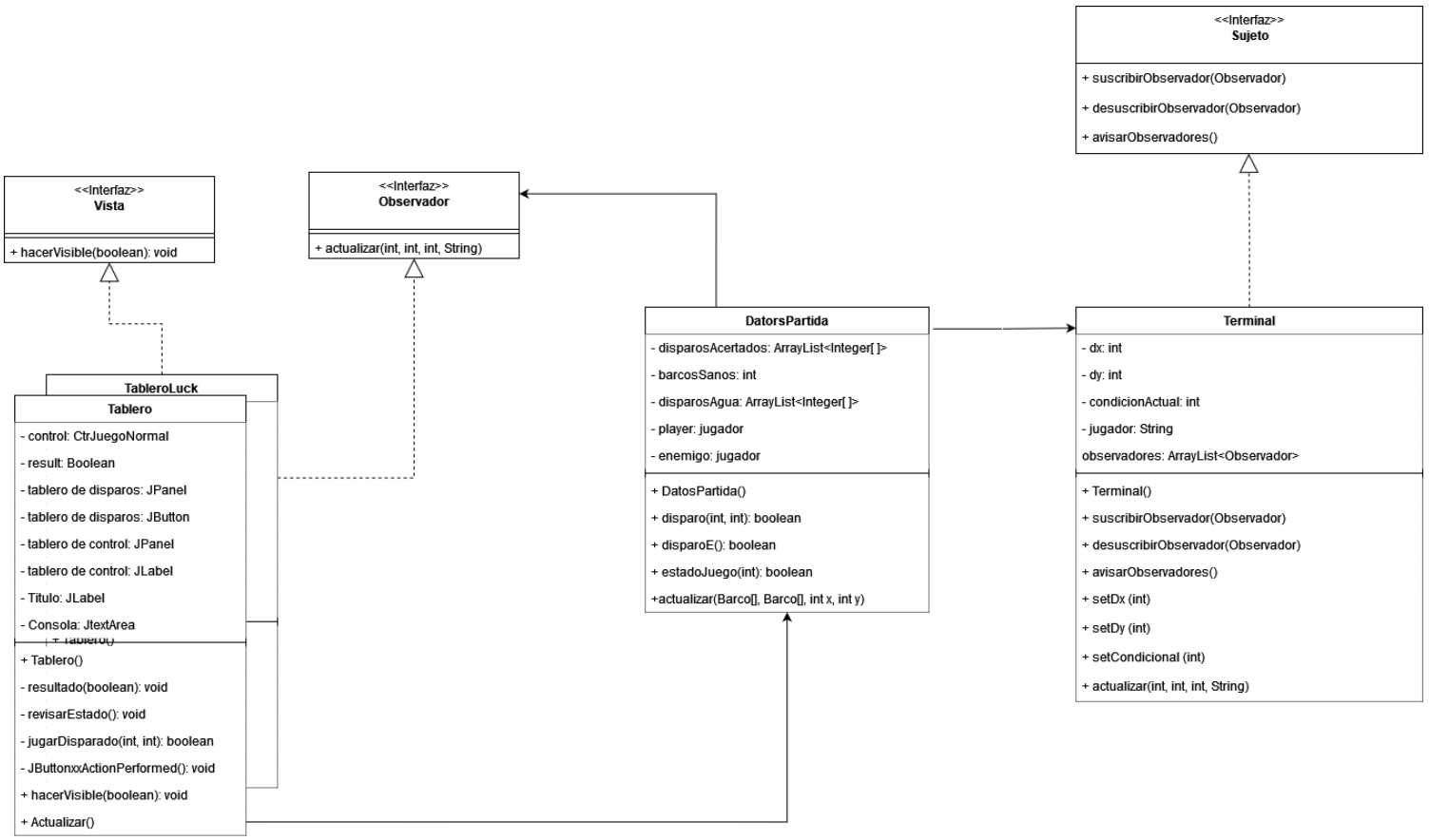

Usuario
Menu
Juego Clasico
Logica
Salir
1: Seleccionar partida clásica
1: Iniciar partida
1.1: Cargar partida
1.2
1.3: Solicitar ubicación barcos
loop
[Barcos posicionados faltantes > 0]
2: Ubicar barcos
2.1: Validar posición
2.2
2.3: Partida lista para empezar
loop
[Barcos enemigos restantes > 0 || Barcos usuario restantes > 0]
3: Disparar posición
3.1: Revisar posición
Alt
[Agua]
3.2: Actualizar animación
[No agua]
Alt
[Barco dañado]
3.2: Actualizar animación
3.3: Actualizar estado del barco
[Barco hundido]
3.2: Actualizar animación
3.3: Actualizar estado del barco
3.4: Actualizar cantidad de barcos enemigos restantes
opt
[Barcos enemigos restantes > 0]
4: Iniciar turno maquina
Finaliza turno maquina
Determinar ganador
Generar mensaje
Mensaje mostrado
Volver al menu
Finalizar partida
Selecciona salir
Salir
Aplicación finalizada

Patrones de Diseño

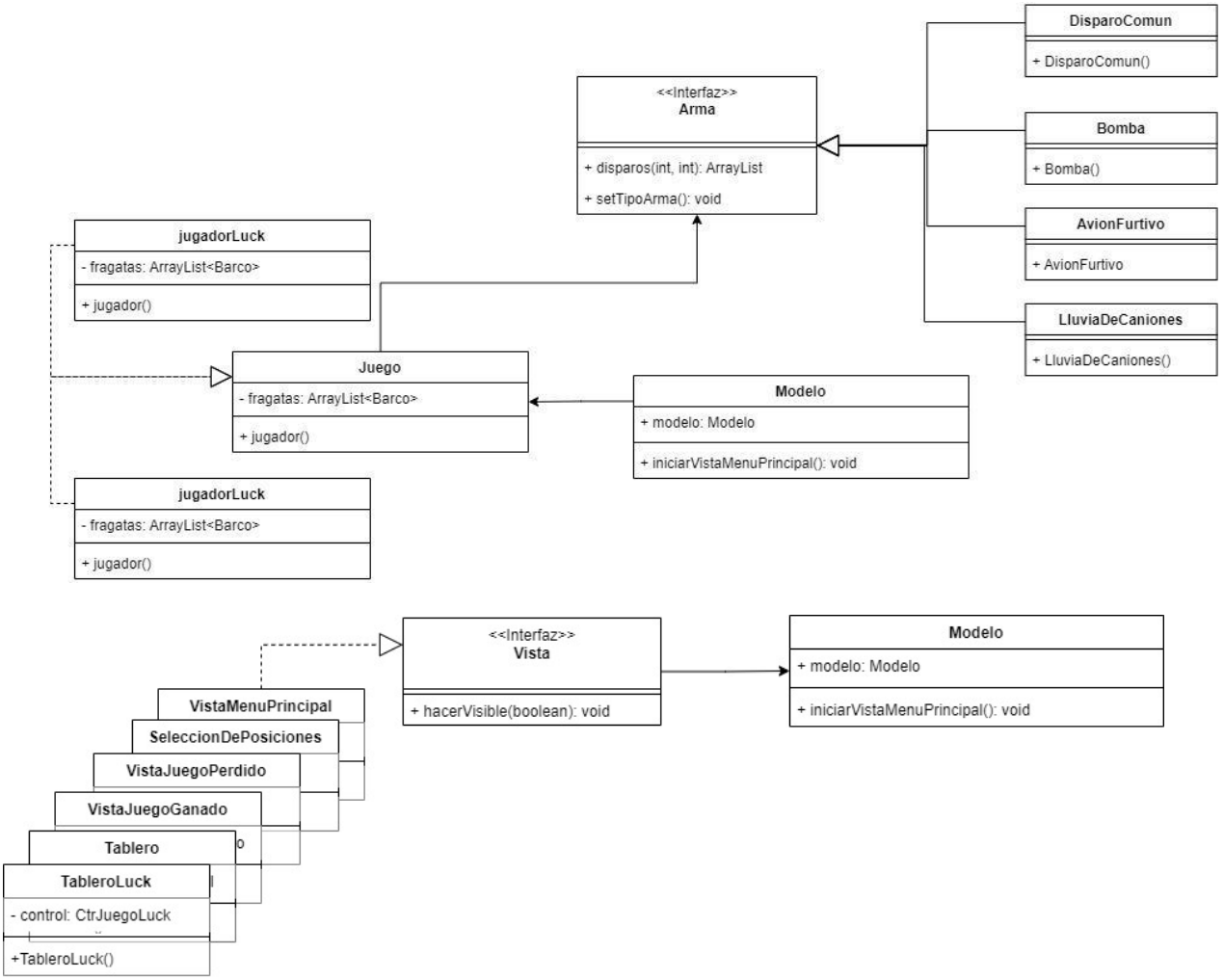
Singleton: Mediante este patrón utilizamos generamos una sola instancia de Modelo a la cual accederemos desde los distintos controladores.



Observer: Mediante este patrón, se actualiza la vista de los tableros del juego (el suscriptor) y, a su vez, se encarga de actualizar la terminal por cada cambio realizado. De esta forma, por medio de la llamada "actualizar", el sujeto Terminal se encargará de llevar la información a las vistas para que puedan actualizarse según los datos enviados. Un ejemplo de ello, es que si se dispara en la posición (0,7), el patrón implementado se encarga de actualizar la casilla según el resultado y de mostrarlo en la terminal de la misma vista.



Strategy: este patrón es utilizado para cambiar entre comportamientos o vistas en tiempo de ejecución. Para nuestro caso lo utilizamos para los distintos modos de juego, donde el tipo de disparos será diferente en ambos casos, así como ciertas condiciones de juego. A parte de tener en cuenta los distintos comportamientos ya especificados, tenemos la posibilidad de agregar nuevos comportamientos a los modos de juego ya existentes, así como agregar nuevos modos sin realizar cambios en la funcionalidad de los anteriormente generados.



Pruebas Unitarias

Las pruebas unitarias permiten comprobar el comportamiento del código a nivel de módulos individuales para asegurar que funcionen correctamente. Aquellas clases que no dependen de otras clases es la clase Barco.

Se presentan a continuación la lista de pruebas unitarias o unit test realizadas:

- BarcoTest: se verifica el funcionamiento correcto de los métodos implementados en la clase, *girar* y *setPosicion*. Se generaron pruebas en cada método para comprobar el correcto funcionamiento ante entradas esperadas y entradas no aceptables.

Pruebas de Integración

Las pruebas de integración corresponden a comprobar el funcionamiento correcto entre módulos. Para ello, se realiza el test de la clase Jugador el cual implementa la clase Barco. Por otra parte, también se comprueba el funcionamiento del patrón Singleton. Entonces:

- jugadorTest: se verifica el funcionamiento correcto de los métodos limitadas a la instanciación de la clase Barco. Los test se limitan al funcionamiento de los métodos donde implementa la clase Barco.

- TestModelo: se genera para comprobar el funcionamiento correcto del patrón Singleton en la clase Modelo. El test simula la existencia de una instancia del Modelo y verifica que al intentar abrir una nueva instancia, sigue existiendo sólo uno.

- DatosPartidaTest: se genera para comprobar el correcto funcionamiento de la lógica de los disparos en el juego, tanto para la máquina como para el jugador. La prueba consiste en revisar si el disparo se almacena en la lista correspondiente y si se recibe alguno de los resultados esperados.

Pruebas de Sistema

Las pruebas de sistema corresponden al funcionamiento del modelo vista controlador donde se comprueba que las entradas esperadas dentro la vistas implementadas, generen la correcta salida debido al correcto enlace entre el controlador correspondiente y el modelo.

Se realizaron los siguientes test de sistema:

- CaseButtonJugarModoClasicoTest: inicia el programa, se inicia el menú principal y se verifica que al seleccionar la opción “Jugar Modo Clásico” la ventana abierta corresponda a la de SeleccionDePosiciones.
- CaseButtonJugarModoLuckShotTest: inicia el programa, se inicia el menú principal y se verifica que al seleccionar la opción “Jugar Modo LuckShot” la ventana abierta corresponda a la de SeleccionDePosiciones.
- CaseButtonSalirTest: inicia el programa, se inicia el menú principal y se verifica que al seleccionar la opción “Salir” el programa se cierre.
- CaseButtonRendirseTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones y se verifica que al presionar el botón “Rendirse”, aparezca una ventana que ofrezca las opciones “Si” y “No”.
- CaseButtonCancelarTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones, se selecciona la opción “Submarino”, se selecciona el casillero “2D” y se verifica que al presionar el botón “Cancelar”, el barco elegido deje de estar seleccionado.
- CaseButtonConfirmarTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones, se selecciona la opción “Submarino”, se selecciona el casillero “2D” y se verifica que al presionar el botón “Confirmar”, se setee el barco en la ubicación.
- CaseButtonIniciarPartidaTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a

SeleccionDePosiciones, se utiliza el método para generar la flota aleatoria de la máquina y se verifica que esta habilitado la opción “Iniciar Partida” y se comprueba el cambio de ventana al de la vista Tablero.

- CaseButtonGirarTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones, se selecciona la opción “Submarino” y se verifica que al seleccionar la opción “Girar”, el barco seleccionado tenga la orientación en vertical.

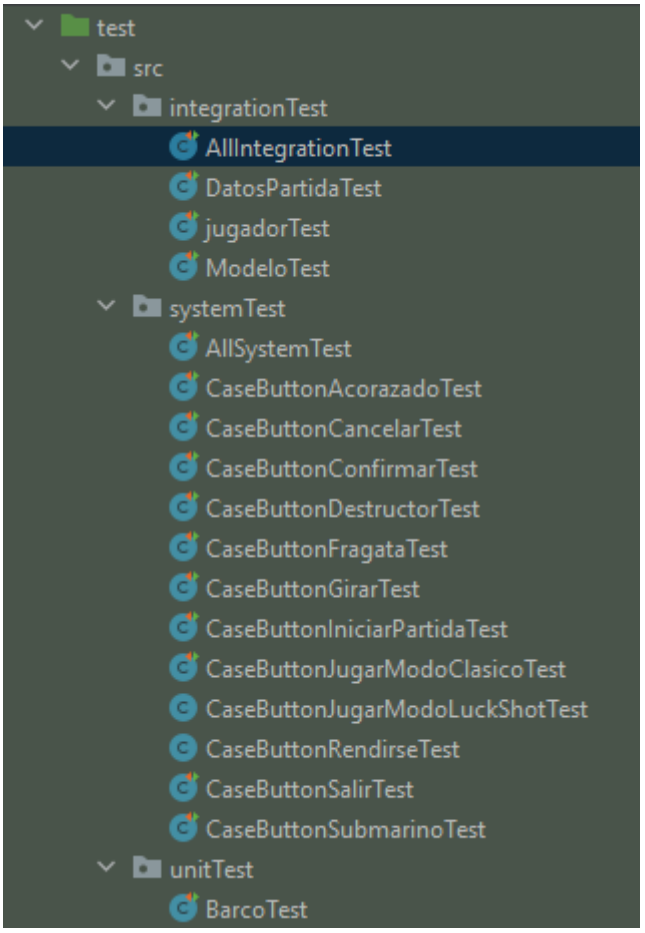
- CaseButtonSubmarinoTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones y se verifica que al seleccionar la opción “Submarino”, el barco seleccionado corresponda a la clase Submarino.

- CaseButtonFragataTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones y se verifica que al seleccionar la opción “Fragata”, el barco seleccionado corresponda a la clase Fragata.

- CaseButtonDestructorTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones y se verifica que al seleccionar la opción “Destructor”, el barco seleccionado corresponda a la clase Destructor.

- CaseButtonAcorazadoTest: inicia el programa, se inicia el menú principal, se selecciona el modo “Jugar Modo Clásico”, se inicia la ventana correspondiente a SeleccionDePosiciones y se verifica que al seleccionar la opción “Acorazado”, el barco seleccionado corresponda a la clase Acorazado.

A continuación, se presenta la lista de pruebas de sistema realizadas.



Implementación

El correcto funcionamiento de los test es requisito para mergear al main el programa realizado desde el branch develop. Para ello, se debe pasar primero por el branch correspondiente a la documentación. Luego, por medio de la integración continua implementada mediante GitHub Actions y Maven, se ejecutan cada uno de los test realizados. El push o pull request no será efectivo hasta que pase cada uno de los test propuestos. El proceso que ejecuta es el siguiente:

- El programa realizado contiene un archivo *pom.xml* el cual instala las dependencias para ejecutar los tests realizados y una carpeta Workflow con el archivo maven.yml el cual da las directrices que deben realizarse ante un pull request o push a la rama main.
- GitHub Actions se encarga de, al momento de intentar hacer un pull request o un push a la rama main, de lo siguiente:
    - Adquiere el programa para tenerlo listo.
    - Configura el JDK según la versión designada y su correspondiente distribuidor.
    - Ejecuta el comando mvn para ejecutar, desde Maven, hasta la fase “test” correspondiente al ciclo de vida o lifecycle del mismo.

Cabe aclarar que, dentro del entorno de desarrollo local, es decir en la IDE donde uno trabaje, se deben correr los tests ya sea de forma individual, por clase de test agrupados o todos en conjunto. Los respectivos archivos (*pom.xml* y *Workflow/maven.yml*) se encuentran dentro de la carpeta del trabajo.