PFC: NavyWar – Capa de presentación

universidad del país vasco

euskal herriko unibertsitatea

Facultad de Informática / Informatika Fakultatea

NavyWar – Capa de presentación

Alumno/a: D.Adrián Gil del Val

Director/a: D.German Rigau i Claramunt

Proyecto Fin de Carrera, junio 2013

Autor: Adrián Gil

PFC: NavyWar – Capa de presentación

Agradecimientos

Antes de empezar con la memoria quería dar las gracias a todas las personas que me han ayudado en la realización de este.
Primero a mi compañero de proyecto Daniel que sin su ayuda el resultado no habría sido el mismo. A mis padres que me han dado su apoyo y me han ayudado en muchos otros aspectos, más de los que ellos creen.
También a mi director de proyecto Germán Rigau Claramunt y al departamento de Lenguajes y Sistemas Informáticos por resolver las dudas que nos surgían y orientarnos durante todo el proyecto.
Y por último a mis compañero de clase Diego, Borja, Asier, Gorka, David, Ion, Tania y Fernando, con los que he compartido estos 4 años de estudios.

Autor: Adrián Gil

PFC: NavyWar – Capa de presentación

Proyecto de Fin de Carrera de Ingeniería Técnica en Informática de Sistemas: NavyWar (Capa de presentación) por Adrián Gil. Director: Germán Rigau. Proyecto conjunto: NavyWar, realizado en colaboración con Daniel Crego (Capa de Negocio y Datos).

Juego desarrollado para dispositivos Android basado en "Hundir la flota". Incluye modos de juego contra inteligencia artificial y para dos jugadores.

Keywords: Juego, Android, Hundir la flota, Colaboración, Presentación

PFC: NavyWar – Capa de presentación

Índice de contenido

Autor: Adrián Gil

PFC: NavyWar – Capa de presentación

1. INTRODUCCIÓN

En este documento se detalla la planificación, gestión y implementación del proyecto fin de carrera (PFC) de la ingeniería técnica en informática de sistemas (ITIS) de la universidad del País Vasco (UPV/EHU): NavyWar (Capa de presentación). Director: Germán Rigau i Claramunt. El proyecto está realizado por Adrián Gil del Val en colaboración con Daniel Crego de la Cal que se encarga del desarrollo de la capa de dominio en el proyecto: NavyWar : Capa de negocio y datos.

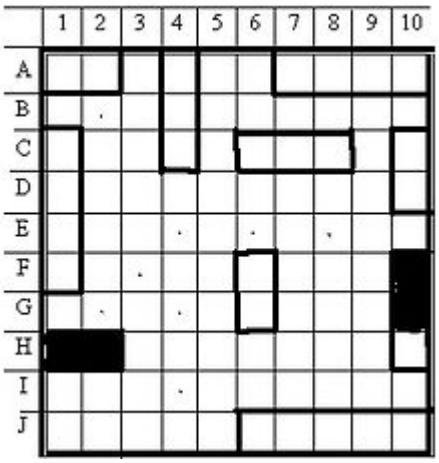


Ilustración 1: NavyWar

NavyWar está desarrollado para dispositivos móviles que funcionen con el sistema operativo Android. Se inspira en el juego *Hundir la flota,* un conocido juego de mesa.

El objetivo de NavyWar es encontrar y hundir la flota del oponente para ello los jugadores disponen de una flota. Cada flota contará con 10 barcos de distinto tamaño. 1 portaaviones, 2 destructores, 3 fragatas y 4 submarinos.

El juego se divide en 2 fases, la fase de preparación y la fase de combate. En la fase de preparación veremos los 10 barcos y un tablero vacío. Tendremos que colocar los 10 barcos en el tablero, dejando siempre un espacio alrededor de cada barco. Una vez colocados todos los barcos empezara la fase de combate.

En esta fase tendremos 2 tablero, el tablero propio, donde veremos los barcos que hemos colocado en la fase anterior, y el tablero rival en el que veremos una niebla que oculta los barcos. En este momento del juego deberemos bombardear las casillas del oponente para encontrar sus barcos. El primero que encuentre y hunda todos los barcos del oponente ganará el juego.

En NavyWar el oponente puede ser o un amigo al que le daremos el móvil cuando sea su turno o una inteligencia artificial (IA). Si es un amigo al acabar cada turno aparecerá una pantalla indicando que debes darle el móvil al oponente. Si es una IA está realiza todas las acciones del juego de forma automática, como colocar los barco o bombardear.

Esta memoria incluye los capítulos: DOP, otros juegos móviles, elección tecnológica, captura de requisitos, análisis, diseño, implementación de la capa de presentación, gestión,pruebas, conclusiones y

anexos. Todos, salvo implementación y conclusiones, son comunes con los del proyecto NavyWar: Capa de Negocio y Datos.

2. DOCUMENTO DE OBJETIVOS DEL PROYECTO

Este proyecto será realizado por los alumnos Adrián Gil y Daniel Crego. Para poder diferenciar las competencias de cada alumno se dividirá en dos partes, la parte gráfica de la que se encargará Adrián Gil y la parte lógica de la que se encargará Daniel Crego.

La parte gráfica se encargará de la gestión de los menús, pantallas, efectos, imágenes y sonidos. Mientras que la parte lógica se encargará de la gestión de los datos, que se cumplan las reglas del juego y de crear una IA capaz de jugar contra el usuario.

2.1. Objetivos

El propósito del proyecto es diseñar, en un plazo de tiempo no muy extenso, un juego para móviles táctiles Android basado en el conocido juego de mesa hundir la flota.
El juego contará con las siguientes especificaciones.

- Dos modos de juego: el juego de para dos jugadores y el juego contra IA que contará con dos dificultades, muy fácil y normal.
- Una pantalla en la que deberá colocar los barcos como él usuario decida, podrá optar entre colocar su flota automáticamente de forma aleatoria o disponerla manualmente barco por barco.
- Una vez colocados los barcos correctamente se le permitirá al usuario empezar el juego cuando el decida.
- Una pantalla en la que se visualiza el tablero rival (sólo las casillas reveladas) sobre este tablero podrá seleccionar una casilla sin revelar y bombardearla o marcarla .
- También podrá visualizar el tablero propio para controlar la situación de su propia flota.

Otro objetivo del proyecto es añadir el juego al mercado de aplicaciones de Android para que puedan disfrutarlo los usuarios de Android.

2.2. Alcance

Hemos diseñado el proyecto de forma modular separándolo en prototipos diferenciados, cada uno de ellos cubre un alcance superior al anterior.

El prototipo I es una partida de dificultad muy fácil contra la IA. Es muy sencilla de contenido. Simplemente se solicita al jugador que coloque su flota en el tablero, luego comienza la partida contra un oponente IA que bombardea de modo aleatorio. Sólo se tendrán en cuenta las reglas básicas como son:

- Al colocar un barco quedará un espacio libre de una casilla de ancho alrededor de este. No se podrá colocar más barcos en este área.
- Cuando se bombardea sobre un barco se tiene otra oportunidad conservando el turno hasta que el resultado sea agua.
- Al hundir un barco las casillas circundantes se revelan automáticamente ya que sólo pueden ser agua.

- Gana el jugador que antes destruya la flota del oponente.

En la parte gráfica el prototipo I se centrará en colores, formas y efectos dejando las imágenes para más adelante.

El prototipo II incluirá mejoras sustanciales respecto a la apariencia y la jugabilidad.

Contará con un menú para seleccionar el modo de juego, dos dificultades para juego contra IA, el modo muy fácil del prototipo I y el modo normal. En el modo normal se tratará de hundir un barco que se haya tocado previamente. Para lograrlo busca en las proximidades ignorando las casillas en diagonal, si de el barco en cuestión se conoce más de una casilla la IA buscará continuando la dirección de dicho barco.

El prototipo II también contará con el modo de juego para dos jugadores en el que ambos jugadores compartirán el mismo dispositivo, primero el Jugador 1 colocará su flota y luego su oponente. Después se elegirá al azar quien empieza a jugar y cada cambio de turno se bloqueará la pantalla para evitar que los jugadores puedan ver la flota de su oponente.

En el prototipo III se incluirá el modo de juego on-line, en el que se podrá jugar una partida contra otro jugador en cualquier momento y en cualquier lugar. Permite tener activas varias partidas a la vez para que no te aburras esperando a que tu oponente haga su movimiento. Ademas podrás elegir a tu oponente o dejar que se te asigne uno de forma aleatoria. También se incluirán mejoras de apariencia, una interfaz y un menú mas intuitivo. Éste prototipo es sin duda el definitivo y tenemos la intención de añadirlo al mercado de aplicaciones de Android. También queremos añadir la posibilidad de ampliar el juego con actualizaciones futuras.

Un ejemplo de actualización sería que el usuario pueda comprar nuevos aspectos. Estos simularán batallas en el espacio, en cielo abierto, en el desierto, en el cementerio de Halloween, en el Polo Norte, bajo el océano, etc.

2.3. Método de Trabajo

2.3.1. Procesos tácticos

Para poder avanzar de forma continua en el proyecto se acordarán unos días a la semana en los cuales los integrantes del grupo se reunirán y pondrán en común lo avanzado individualmente. Se hablará sobre lo siguientes pasos a realizar en el proyecto así como comentar los problemas encontrados desde la ultima reunión y proceder a su corrección. Como los integrantes del grupo tienen su residencia en distintas ciudades se vuelve complicado realizar reuniones de forma habitual. Acordamos realizar las reuniones vía Skype que nos permitirá estar en contacto. Aun contando con el uso de esta herramienta se realizarían reuniones de forma física en San Sebastián periódicamente.

2.3.2. Procesos operativos

Para la parte operativa se decidió dividir el proyecto en dos partes la gráfica y la lógica. Cada miembro se ocupará de una de ellas aunque podamos colaborar en la otra parte en cualquier momento. La parte lógica se encargará de gestionar las reglas del juego como por ejemplo los turnos de los jugadores, la

inteligencia artificial o el bombardeo de casillas. La parte gráfica se encargará de las distintas pantallas del juego, de la interacción con el usuario y de indicar a la parte lógica cuándo debe de actuar. Ambas partes estarán muy relacionadas.

2.3.3. Representantes

Dado que sólo somos dos componentes hemos decidido ser co-directores de modo que el trabajo de gestión lo realizamos en conjunto y por consenso.

2.3.4. Recursos disponibles

Para coordinar los esfuerzos a la hora de desarrollar el proyecto es necesario saber de qué recursos podemos disponer:

Recursos humanos

Para la elaboración del proyecto disponemos de un grupo formado por dos personas, Adrián Gil y Daniel Crego.

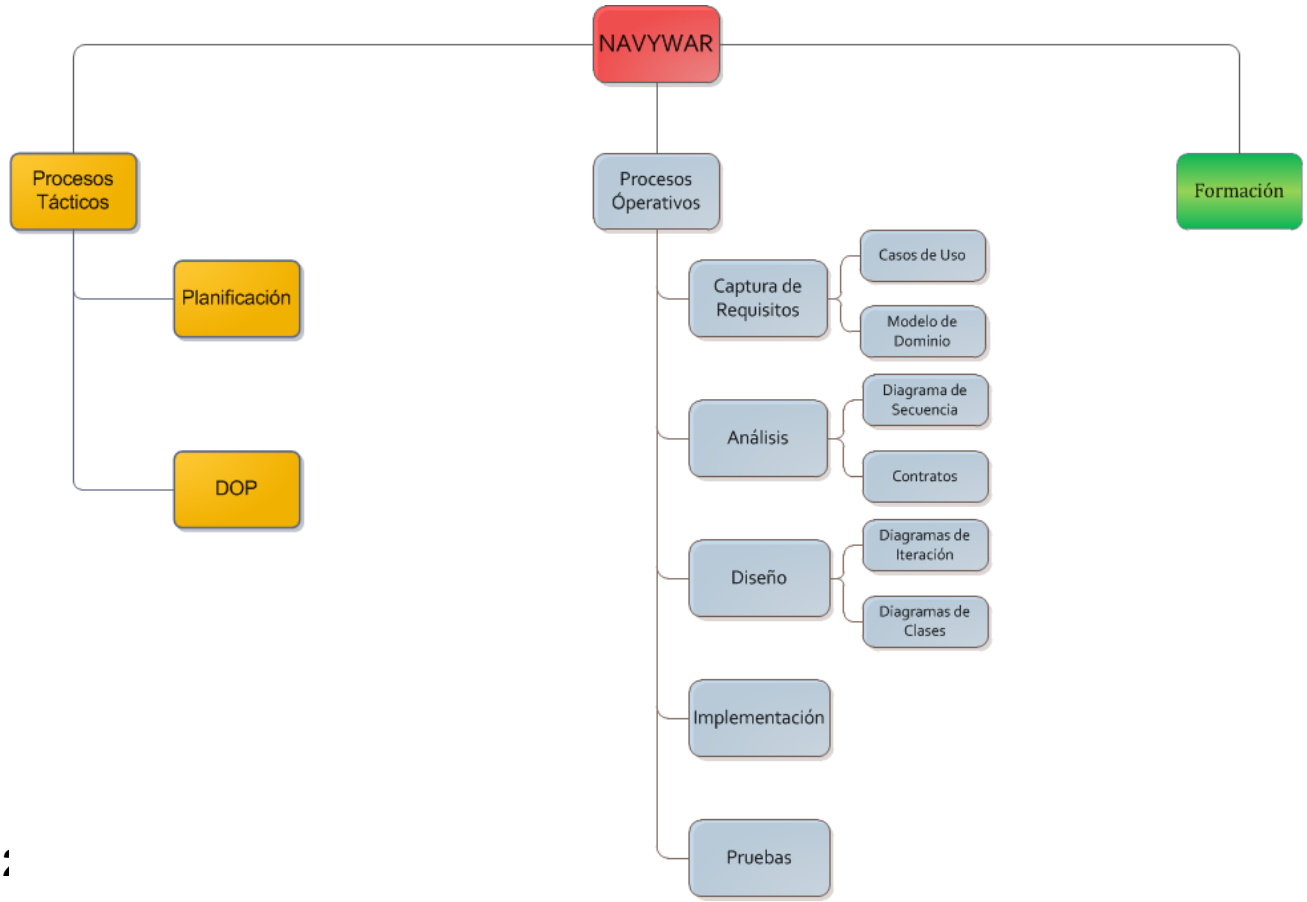
Recursos materiales

Cada miembro del grupo dispone de:

- Cómo mínimo un ordenador con conexión a Internet con el que trabajar.
- El software necesario para llevar a cabo el proyecto.
- La universidad dispone de numerosas instalaciones útiles como espacio para organizar reuniones.
- Una biblioteca en la que encontrar diversos libros si se diera la necesidad.

PFC: NavyWar – Capa de presentación

Diagrama EDT



Para gestionar la evolución del proyecto utilizaremos un proceso de desarrollo iterativo e incremental. Dividiremos el proyecto en una serie de prototipos para usarlos como punto de referencia.

0 -	1/11/2012	Prototipo I	– Colocación de barcos y juego
1 -	19/11/2012	Prototipo II	– Mejora gráfica, menú y modos de juego (IA)
2 -	21/1/2013	Prototipo III	– Mejora de menú, modos de juego, gráfica y modo on-line
3 -	21/2/2013	Memoria	– Realización de la documentación del proyecto

Prototipo I – Colocación de barcos y juego

En esta entrega empezaremos con las fases del juego de preparación y de combate.

En este momento el aspecto gráfico del juego sera mínimo, nos centraremos en la parte lógica.

Prototipo II – Mejora gráfica, menú y modos de juego (IA)

A partir de esta entrega empezaremos a incluir imágenes y movimientos para los gráficos del juego. También mejoraremos la parte lógica y incluiremos distintas IA así como un modo de dos jugadores.

PFC: NavyWar – Capa de presentación

Prototipo III – Mejora de menú, modos de juego, gráfica y modo on-line

En esta entrega mejoraremos todo lo anterior y intentaremos crear un modo de juego para dos jugadores vía web.

2.4. Planificación Temporal

2.4.1. Procesos tácticos

Gestión

Subtareas			Duración	Persona	Esfuerzo
Desarrollo del DOP	Introducción y objetivos		1h	Adrián	1h
	Alcance	Recursos disponibles	3h	Ambos	6h
		Lista de entregas			
		Lista de subtareas			
		Diagrama EDT			
		Asignación de recursos			
	Planificación temporal	Tareas	3h	Daniel	3h
		Diagrama de Gantt			
	Plan de contingencia		3h	Daniel	3h
	Método de trabajo	Tácticos	2h	Adrián	2h
		Operativos			
		Formativos			
	Estimación de costes		1h	Adrián	1h
Gestión de archivos		Compartición Web	2h	Ambos	4h
		Copias de seguridad			

Reuniones

Subtareas	Duración	Persona	Esfuerzo
DOP	3h	Ambos	6h
Captura de Requisitos	2h	Ambos	4h
Análisis	3h	Ambos	6h
Diseño	4h	Ambos	8h

2.4.2. Procesos operativos

Desarrollo

Subtareas			Duración	Persona	Esfuerzo
Captura de Requisitos	Modelo de casos de Uso	Casos de uso	4h	Ambos	8h
		Diagrama de casos de uso			
	Modelo de dominio		1h	Ambos	2h
Análisis	Diagrama de secuencia		2h	Ambos	4h
	Contratos		2h	Ambos	4h
	Modelo conceptual		1h	Ambos	2h
Diseño	Diagrama de iteración		5h	Ambos	10h
	Descripción		4h	Ambos	8h
Implementación			100h	Ambos	200h

Reuniones

Subtareas	Duración	Persona	Esfuerzo
DOP	3h	Ambos	6h
Captura de Requisitos	5h	Ambos	10h
Análisis	6h	Ambos	12h
Diseño	14h	Ambos	28h
Implementación	30h	Ambos	60h

2.4.3. Procesos formativos

Subtareas	Duración	Persona	Esfuerzo
Investigación y reuniones	15h	Ambos	30h

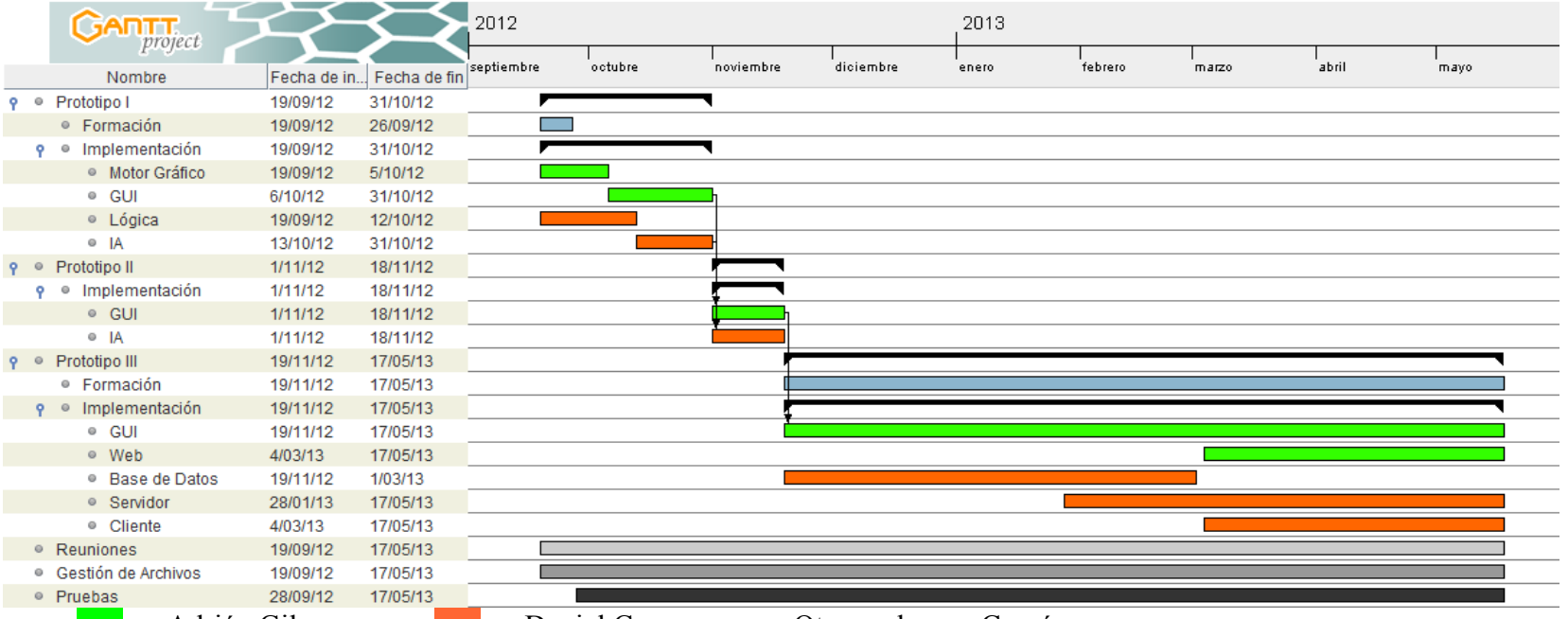
2.4.4. Totales

El esfuerzo del alumno Adrián Gil es de 213 horas.

El esfuerzo del alumno Daniel Crego es de 215 horas.

Con los datos anteriores la duración total del proyecto es 221 horas con un esfuerzo total de 428 horas.

2.4.5. Diagrama de Gantt



2.5. Plan de Contingencia

En toda buena planificación es vital adelantarse a los posibles contratiempos y desarrollar unas medidas de contención adecuadas a las necesidades y recursos del proyecto.

2.5.1. Baja Temporal de un miembro

Relevancia: ★★* Probabilidad: Alta

Si un componente del grupo sufriera algún tipo de baja incapacitante temporalmente su carga de trabajo se desplazaría a más adelante o se compartiría con otro miembro.

*Esta incidencia tiene una relevancia mayor cuanto más tiempo tome efecto.

2.5.2. Baja Permanente de un miembro

Relevancia: ★★★★★ Probabilidad: Baja

Si un componente del grupo sufriera algún tipo de baja incapacitante de forma permanente su carga de trabajo no se podría desplazar en su totalidad a otro miembro, de forma que se realizaría un replanteamiento y una reducción del Alcance. Si el grupo desapareciese el proyecto quedaría congelado indefinidamente.

2.5.3. Pérdida del trabajo realizado

Relevancia: ★★★★ Probabilidad: Baja

Ante la posibilidad de una pérdida parcial o total del trabajo realizado será necesario mantener asiduamente copias de respaldo en varios sistemas independientes, algunos de ellos aislados.

2.5.4. Inutilización de los medios de desarrollo

Relevancia: ★* Probabilidad: Baja

Para enfrentar la posible pérdida o incapacitación de algún medio imprescindible para el correcto desarrollo del proyecto sería vital sustituir la pérdida con prontitud y si es posible utilizar medios alternativos mientras dure la reposición.

*Esta incidencia tiene una relevancia mayor cuanto más tiempo tome efecto.

2.5.5. Desviación de los plazos establecidos

Relevancia: ★★ Probabilidad: Alta

Se tratará de seguir el plan estipulado lo más fielmente posible. Si la previsión indica que no será posible se recortaría el alcance al último prototipo estable o se ajustarían los plazos según convenga.

2.5.6. Declive de la plataforma elegida

Relevancia: ★★★ Probabilidad: Baja

El código habrá de ser modular y correctamente estructurado de modo que sea relativamente sencillo proceder a una portabilización en caso de detectar un serio declive en la plataforma seleccionada como base para el proyecto (Smartphones, Android).

2.6. Programas utilizados

Para la gestión de documentos necesitaremos estas herramientas.

2.6.1. Eclipse

Esta plataforma de desarrollo de software ofrece un entorno de desarrollo Android por lo que nos resultará muy útil para desarrollar el código. Ademas ambos desarrolladores conocemos esta herramienta.

2.6.2. LibreOffice

Para documentar el proyecto utilizaremos LibreOffice. Esta conocida plataforma de ofimática es totalmente gratuita y ofrece múltiples herramientas, como hojas de calculo o edición de gráficos.

2.6.3. Google Drive

Si en algún momento necesitamos editar documentos de forma concurrente utilizaremos Google Drive. Este servicio de Google nos ofrece 5 GB de almacenamiento y distintos tipo de fichero como diagramas, dibujo o documentos de texto.

2.6.4. Skype

Al ser inviable para los integrantes del proyecto realizar reuniones de forma física utilizaremos una forma de hacerlo por internet. Skype es uno de los sistemas de comunicación web más utilizados y permite otras funcionalidades como el envío de archivos, uso de webcam o compartir el escritorio. Herramientas muy útiles para facilitar todo tipo de intercambios. Por estas y otras razones y ya que ambos miembros contábamos con una cuenta de Skype usaremos este programa.

2.6.5. TortoiseSVN

Necesitaremos una forma de compartir el código fuente de la aplicación de una forma eficiente. Hemos consultado las distintas formas en las que se puede compartir y hemos encontrado un cliente de *SVN* llamado *TortoiseSVN* y el servicio de *Google Code* que te permite tener un repositorio con el código.

2.6.6. SolidWork

Este programa se utilizará para crear imágenes. Aunque no esta orientado a la creación de imágenes en 2D nos permitirá darle un aspecto más atractivo al juego.

2.6.7. Photoshop CS3

Para la edición de imágenes utilizaremos photoshop este programa nos permite retocar imágenes aplicándoles capas o modificando pixeles para darles el aspecto deseado.

2.7. Factibilidad

Pese a que no contamos con todos los conocimientos de Android necesarios, si que contamos con conocimientos en Java. Ademas tenemos mucha iniciativa y ganas de aprender así como la capacidad de investigación, cualidades que nos serán muy útiles durante este proyecto.

También disponemos de las herramientas necesarias como un entorno de programación, comunidades de usuarios para la consulta de dudas, estas son indispensables para finalizar este proyecto con éxito.

Por lo tanto con los datos presentados anteriormente y las ganas de los integrantes pensamos que la realización de este proyecto es posible.

3. OTROS JUEGOS MÓVILES

Existe una gran variedad de juegos para smartphones. A continuación detallamos algunos juegos que consideramos relevantes.

3.1. Juegos Móviles

Los smartphones se han convertido en la plataforma de juegos del momento, son capaces de competir con consolas portátiles en jugabilidad y calidad gráfica. Los sistemas más utilizados en estos dispositivos son Android y iOS. Ambos cuentan con gran cantidad de juegos de alta calidad, muchos de ellos comunes como *Apalabrados*, *Angry Birds* o *Temple Run.*

Entre los juegos de móviles encontramos gran variedad de categorías. Juegos de acción, de carreras, juegos para la mente y muchos más.

Algunos utilizan el acelorómetro para detectar los movimientos del móvil y así añadir jugabilidad a los juegos, por ejemplo en los juegos de carreras utilizar el móvil como volante, esto unido a la pantalla táctil permite crear juegos muy interactivos.

3.2. Juegos similares

Cuando empezamos a desarrollar este juego apenas existían juegos similares. El mas parecido era *hundir la flota* de *Lokthrow* (Ilustración 1) el cual no nos parecía que tuviera una gran calidad gráfica ni de IA, no lo consideramos un juego altamente avanzado.

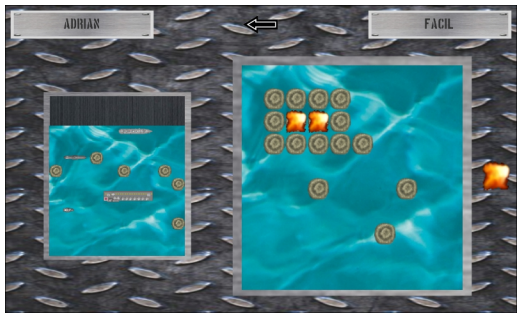


Ilustración 1 - Lokthrow

Un tiempo después de empezar con el juego apareció otro juego basado en hundir la flota, *Hundir la flota* de *MMV* (Ilustración 2) el cual aunque era una mejora considerable respecto a el anterior mantenía una calidad gráfica baja.

Ilustración 2 - MMV

Sin embargo cuando ya estábamos prácticamente acabando el proyecto descubrimos otro juego, *BattleFriends at Sea* de *Tequila mobile* (Ilustración 3) este juego cuenta con gráficos en 3D de gran calidad, juego on-line y redes sociales por lo que puede presentar una alternativa a nuestro desarrollo.

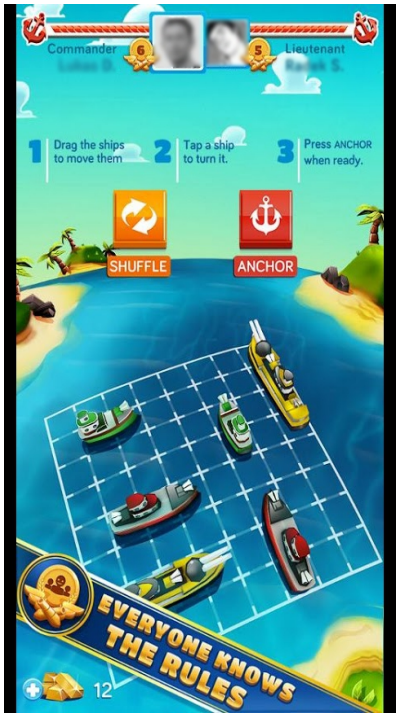


Ilustración 3 - Tequila mobile

4. ELECCIÓN TECNOLÓGICA

En este apartado explicaremos que motivos nos han movido a seleccionar cada uno de los aspectos de este proyecto.

4.1. Android

Elegimos esta plataforma para la distribución de nuestro proyecto por que el desarrollo de software para smarphones está actualmente en auge. Eso nos llevó a interesarnos por saber un poco más sobre este sistema. Después de informarnos sobre *Android* descubrimos que ofrece bastante soporte para desarrolladores y que es relativamente sencillo añadir tu aplicación al *Play store* de *Android* para que pueda descargarla cualquier usuario. Sabiendo esto y ya que ambos disponíamos de dispositivos *Android* nos decantamos por esta plataforma.

4.1.1. Versiones Android

Cada cierto tiempo Android saca nuevas versiones de su sistemas ademas de que cada dispositivo funciona con una versión de este. Las 1.x y 2.x son para dispositivos móviles y las versiones 3.x son para tabletas pero las últimas que han aparecido las 4.x sirven para ambos dispositivos, éstas no están ligadas a un dispositivo ya que es la propia empresa la que decide si sacará pública una versión para el dispositivo.

Por eso a la hora de desarrollar el software tendríamos que elegir si lo desarrollamos para una versión de las mas utilizadas actualmente o para una de las últimas que han salido al mercado, normalmente si funciona para una versión antigua funcionará también para una versión mas moderna por eso lo nos decidimos por elegir la versión 1.5 que es una de las más extendidas y así tendremos la certeza de que será funcional para la mayoría de versiones actuales así como para versiones futuras de Android.

4.1.2. Juegos Android

Una vez decidido que nuestro proyecto se desarrollaría sobre *Android* nos llego la cuestión de qué tipo de aplicación podíamos desarrollar. La tienda de *Android* cuenta con gran cantidad de aplicaciones de muchos tipos desde informativas, juegos, de lectura, de dibujo, de cálculo... La mayoría de estas aplicaciones necesitan un servidor red para actualizar sus datos lo cual consideramos interesante. En un principio consideramos los juegos on-line como una opción remota de desarrollo pero la ampliación de la duración del proyecto nos ha brindado la posibilidad de indagar en esta opción.

4.2. Java

Después nos tocaba elegir que sobre que lenguaje lo realizaríamos. Para *Android* existen dos lenguajes de programación, *Java* y *C++*. Entre estos dos lenguajes el que más conocemos es *Java* que lo hemos utilizado en múltiples asignaturas así que la selección estaba clara. Además tras informarnos descubrimos que la mayoría de aplicaciones están desarrolladas en *Java* y que muchos desarrolladores

aseguran que el desarrollo de juegos es más fácil en *Java*.

4.3. Alternativas tecnológicas de la capa de presentación

Ha continuación se expondrán y explicarán algunas de las alternativas disponible para la capa de presentación.

4.3.1. Android Widgets

Las opciones que te ofrece Android para desarrollar aplicaciones son muy amplias y ofrecen distintas opciones desde listas, botones, scroll, estilos, animaciones y otros widgets pero todas estas opciones están orientadas a otro tipo de aplicaciones, no a juegos. Para este tipo de aplicaciones apenas ofrecen opciones por lo que utilizaremos estas funciones para algunas partes del juego como menús o listas pero no para el juego principal.

4.3.2. Canvas Android

Finalmente nos decidímos a utilizar la clase Canvas de Android para realizar las acciones de pintado y modificación de imágenes junto con alguna adaptación de las pantallas de Android, un sistema de actualización y pintado por capas utilizando actores y grupos de actores que se explicarán más adelante. Utilizando este sistema se ha logrado un resultado bastante elegante para los gráficos.

5. ARQUITECTURA

El proyecto está separado en dos capas. La capa de presentación que interactúa con el usuario mediante mensajes e imágenes. Y la capa de dominio que, mediante una lógica, gestiona los datos que se le entregan a la capa de presentación.

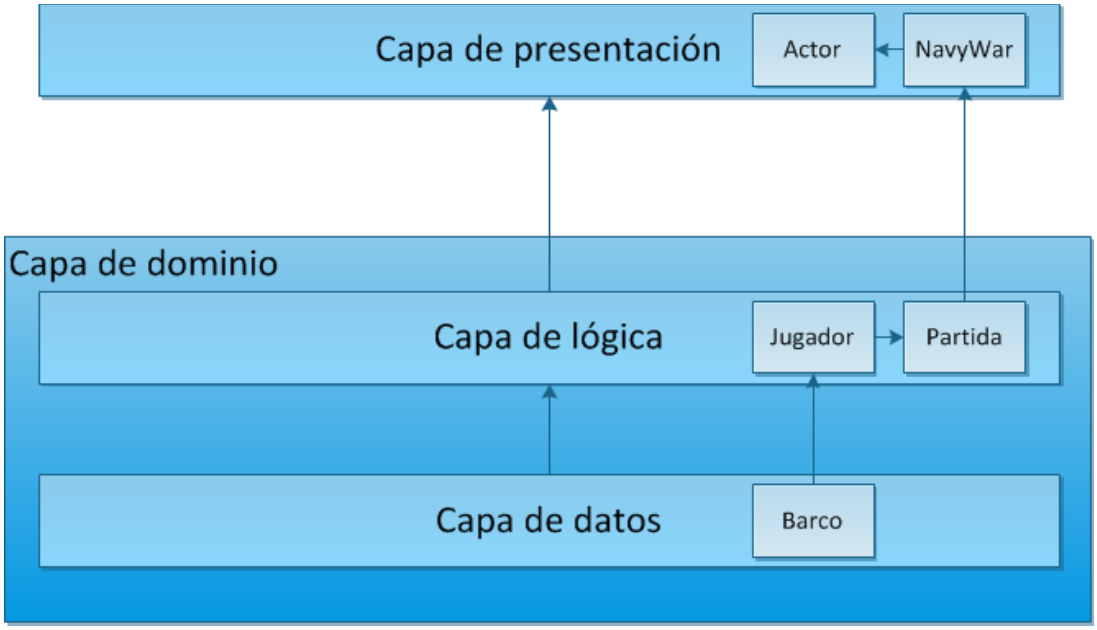


Ilustración 5: Arquitectura

Al estar planificado por prototipos la arquitectura del proyecto evolucionó en cada uno de ellos. A continuación explicaremos como se han sucedido los cambios.

5.1. Prototipo I

5.1.1. Capa de presentación

En este prototipo hemos desarrollado los controladores de pantalla y de gráficos además de la estructura de actores.

Controladores

Para interaccionar con el usuario utilizamos controladores de Android a los que hemos añadido funcionalidades adaptándolos a nuestras necesidades.

Controlador de entrada

Este controlador de entrada se encarga de recibir los toques producidos en la pantalla. Estos pueden ser de 3 tipos, pulsar, levantar y arrastrar. El controlador nos proporciona tanto el tipo como la posición de estos toques y los almacena en una lista para su consulta cuando sea necesaria.

Otro aspecto que se controla es si se ha pulsado alguna tecla del dispositivo. Android proporciona funciones para controlar las principales teclas del dispositivo como el botón back o el botón de opciones.

Controlador de gráficos

En este prototipo el controlador gráfico se encarga de la gestión de formas y colores. Para ello utiliza los colores y las formas que te ofrece la librería graphics de Android.

Actor y ActorGroup

La estructura actor - actorgroup se utiliza para simplificar la asignación de responsabilidades de cada elemento gráfico siguiendo una jerarquía en árbol.

Los actores representan a un elemento del juego y están dentro de un grupo que se encarga de gestionarlos. De esta forma si le indicamos a un grupo que realice una acción tendrá efecto en sus actores. Con este sistema conseguimos organizar los elementos del juego de una forma lógica y ordenada.

5.1.2. Capa de dominio

Estructura y gestión de datos

Para gestionar los datos utilizamos una estructura orientada a objetos. Según este sistema cada elemento del juego tiene una representación en el modelo de datos. Así logramos un acceso rápido a los datos y una gestión eficiente y lógica.

IA muy fácil

La capa de dominio ofrece una interfaz a la capa de presentación. En el caso de una partida contra IA la interfaz es utilizada directamente por el propio sistema. La IA muy fácil bombardea aleatoriamente cuando es su turno.

5.2. Prototipo II

5.2.1. Capa de presentación

Controlador gráfico

A partir de este prototipo el controlador se encarga de gestionar imágenes. Usaremos 3 formatos de imagen RGB_656, ARGB4444 y ARGB8888. El formato rgb no acepta transparencias por eso normalmente utilizamos argb la diferencia entre los dos es la calidad de cada formato argb8888 utiliza mas espacio para la imagen pero esta tiene más calidad.

Los textos también se administran en el controlador. Hemos utilizado un estilo de letra que pensamos se adapta al estilo del juego. Para los textos necesitaremos un tamaño de letra, un texto y un color. El tamaño del texto puede resultar un problema ya que cada dispositivo tiene unas dimensiones diferentes. Para solucionar este problema se ha añadido una función al controlador que al darle un texto y unos píxeles nos calcula que tamaño de letra debe tener el texto para ocupar el espacio correspondiente.

El ultimo funcionamiento del que se encarga el controlador es el clipping. El clipping es el proceso por el que se definen una zona sobre la que los píxeles no deben sufrir ninguna modificación, en los procesos de dibujo que se realizan mientras este activo. Esto nos permite abrir espacios en algunas partes como se muestra en la siguiente imagen.

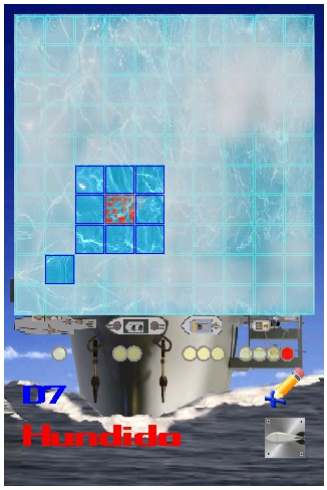


Ilustración 6 - Clipping

En la Ilustración 6 se puede ver cómo se usa el clipping para abrir un espacio en la niebla y así revelar las casillas ya bombardeadas.

Controlador de Audio

Este controlador se encargará de gestionar los archivos de audio. Para ello cuenta con las funciones clásicas de reproducción, pausa y stop. Para la reproducción de sonido se ha incluido la opción de reproducir sonidos de forma cíclica, consiguiendo así un sonido permanente típico de los juegos.

Como esta clase utiliza clases nativas de Android no presenta ningún problema por el formato del archivo de audio que se reproduzca.

Menús Android

Para crear pantallas Android se utilizan los Android Widgets hay que crear unos archivos xml donde especificamos que elementos vamos a utilizar en esa pantalla. De esta forma se separa el contenido y la funcionalidad de los elementos de esa pantalla. Utilizamos estas pantallas para los menús de inicio y de pausa, ya que están más preparadas para este cometido.

5.2.2. Capa de dominio

IA Normal

En este prototipo se especifica otra IA con un nivel mejorado en el que al tocar un barco intenta hundirlo.

6. CAPTURA DE REQUISITOS

En este capítulo se describe el juego y la interfaz. Se incluyen el Modelo de Casos de Uso (MCU), los casos de uso extendidos con descripción y el Modelo de Dominio (MD).

6.1. Descripción de la interfaz de usuario

La interfaz del sistema consta de un menú que da acceso a los distintos tipos de juego. Una vez iniciado el juego se muestra una pantalla para colocar los barcos de diferentes maneras. Tras colocar los barcos se puede iniciar la fase de combate en la que se muestran dos tableros y el progreso del juego. Cada pantalla dispone de un menú acorde al estado de la partida.

6.2. Descripción del juego

Se trata de un juego para dos jugadores. Cada uno dispone de un tablero de 10x10 casillas y 10 barcos:

- 1 portaaviones de 4 casillas.
- 2 destructores de 3 casillas cada uno.
- 3 fragatas de 2 casillas cada una.
- 4 submarinos de 3 casillas cada uno.

El sistema permite al usuario elegir entre tres opciones. Las dos primeras son contra IA con dificultades "muy fácil" y "normal". La tercera es un modo para dos jugadores que comparten el mismo dispositivo.

El juego constará de dos fases. La primera es la fase de preparación en la que los jugadores colocan los barcos. La segunda es la fase de combate en la que los jugadores bombardean el tablero de su oponente por turnos.

En la fase de preparación los jugadores pueden colocar los barcos a voluntad, para ello se le permite girar, mover y quitar los barcos. Además se pueden colocar los barcos no colocados de forma aleatoria o recoger todos los colocados. En cualquier caso al colocar varios barcos siempre deberá disponer de al menos una casilla de separación entre dos cualesquiera y en todas las direcciones, incluso en diagonal. Una vez que ambos jugador decida que todos sus barcos están bien colocados dará comienzo la fase de combate.

En esta fase el sistema le muestra al jugador dos tableros, el propio con los barcos colocados y las acciones del enemigo y el tablero rival en el que al principio no hay nada visible. El jugador podrá seleccionar una casilla y en ese momento se le indica qué casilla ha seleccionado. Una vez seleccionada puede bombardearla. Al hacerlo se le muestra cuál ha sido el resultado obtenido y dependiendo si ha acertado en un barco o no, podrá seguir bombardeando casillas. En este momento del juego también se le permite al jugador marcar una casilla o desmarcarla si ya estaba marcada, una casilla marcada no puede ser bombardeada. Cuando el usuario bombardee una casilla y el resultado sea agua puede pasar el turno a su oponente.

El oponente puede ser otro jugador o el sistema mismo. En el primer caso, al jugador se le muestra una pantalla de espera indicando que el jugador debe entregarle el dispositivo a su oponente. Cuando el oponente pulse la pantalla desaparecerá el mensaje de espera y se habrán cambiado los tableros adaptándose al nuevo jugador. En el caso de que el oponente sea el sistema, la IA actuará según la dificultad que se le halla indicado al iniciar la partida.

6.3. Actores

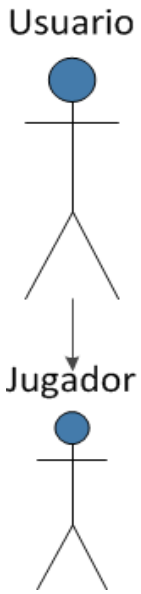


Ilustración 7 – jerarquia de actores

Los actores que tomarán parte en nuestra aplicación se encuentran divididos en varios grupos según el rol que toman al interactuar con el sistema, véase ilustración 7.

El usuario tiene la capacidad de iniciar y terminar la aplicación, así como de crear partidas.

Una partida tendrá dos jugadores. Uno de ellos es el que previamente llamamos usuario. El otro es controlado por otra persona o por el propio sistema según el modo de juego.

6.4. Modelo de Casos de Uso

A continuación se describen los casos de uso agrupados por su actor principal.

6.4.1. Casos de Uso del Usuario

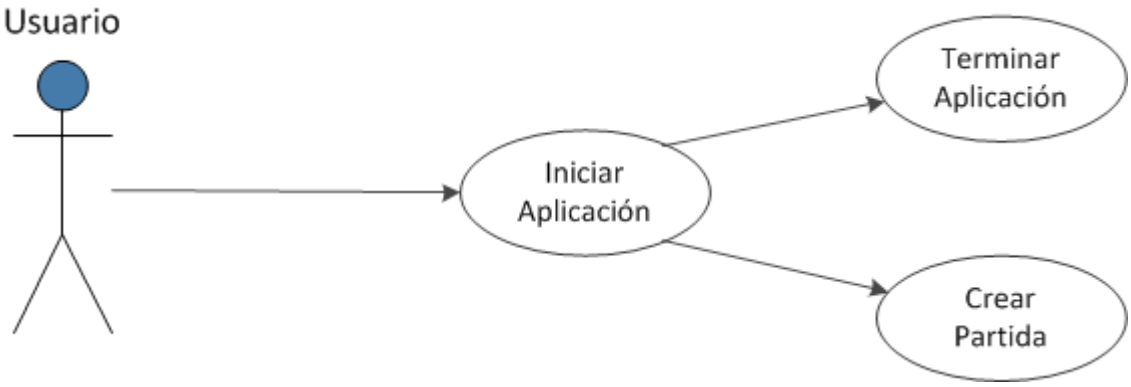


Ilustración 8 – Casos de uso del usuario

En la ilustración 8 se muestran los casos de uso del usuario. El usuario va a tener la posibilidad de iniciar una serie de casos de uso con el sistema. A saber, Iniciar la aplicación (Iniciar Aplicación), salir de la aplicación (Salir) y crear una nueva partida entre los distintos tipos de juego que existen (Crear Partida). Además el usuario en cualquier momento puede salir del sistema (Salir).

Caso de Uso: Iniciar Aplicación

Actores:
Usuario

Descripción:
El usuario inicia la aplicación, el sistema carga todos los datos necesarios y muestra el menú principal.

Precondiciones: -

Postcondiciones:
El sistema muestra un menú interactivo que permite crear una partida, mostrar la lista de partidas activas, mostrar la información de ayuda, o salir.

Escenario principal:
1. **Usuario**: Lanza la aplicación.
2. **Sistema**: Muestra el menú principal.
3. **Usuario**: Selecciona la opción crear partida.
4. **Sistema**: Continua con el C.U. Crear Partida.

Curso alternativo:
Paso 2: Se requieren datos adicionales.
1. **Sistema**: Carga los datos necesarios.

Paso 5: El usuario selecciona la opción de salir.
1. **Sistema**: Inicia el Caso de uso: Terminar Aplicación.

PFC: NavyWar – Capa de presentación

Caso de Uso: Terminar Aplicación

Actores:
Usuario

Descripción:
En cualquier momento el usuario puede terminar la aplicación.

Precondiciones:
Aplicación iniciada.

Postcondiciones:
El sistema muestra un mensaje pidiendo confirmación. Si el usuario acepta la aplicación terminará.

Escenario principal:

1. **Sistema:** Muestra el mensaje de advertencia.
2. **Usuario:** Acepta.
3. **Sistema:** Termina la aplicación.

Curso alternativo:
Paso 2: El usuario no acepta.

1. **Sistema:** Termina el Caso de uso sin cambios.

Caso de Uso: Crear Partida

Actores:
Usuario

Descripción:
En este caso de uso el usuario decide iniciar una partida nueva, el sistema debe mostrarle los distintos modos de juego. El usuario elige el que el prefiera y se crea una nueva partida de acuerdo a lo seleccionado. Después de esto se iniciara la fase de preparación.

Precondiciones: -

Postcondiciones:
Se ha iniciado una nueva partida.

Escenario Principal:

1. Usuario: Pulsa el botón de nueva partida.
2. Sistema: Muestra una lista de modos.
3. Usuario: Escoge un modo.
4. Sistema: Crea la partida en el modo correspondiente e inicia el C.U. Colocar Flota.

Curso Alternativo:
Paso 3: El usuario pulsa el botón de volver

1. **Sistema:** Vuelve al menú principal.

6.4.2. Casos de Uso del Jugador

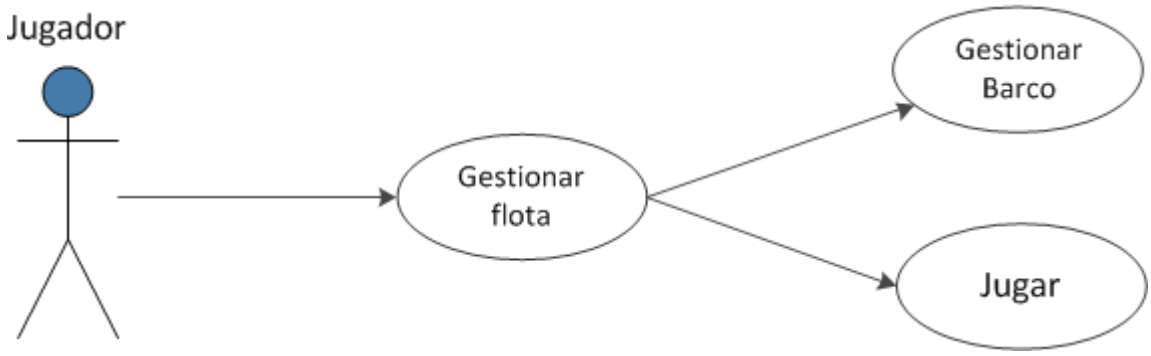


Ilustración 9 – Casos de uso del jugador

En la ilustración 9 se muestran los casos de uso del jugador. El jugador va a tener la posibilidad de iniciar una serie de casos de uso con el sistema una vez que la partida esté comenzada. A saber: colocar, recoger o girar todos los barcos(Gestionar Barcos); colocar, recoger o girar el barco seleccionado (Gestionar Barco Seleccionado); bombardear una casilla y finalizar su turno (Jugar).

Caso de Uso: Gestionar Flota

Actores:
Jugador

Descripción:
En este caso de uso el jugador controla su flota de barcos: puede seleccionar un barco, colocar aleatoriamente los barcos y girar o retirar los barcos colocados.
Este caso de uso dura mientras queden barcos sin colocar, cuando estén todos colocados el jugador indicará que quiere comenzar la fase de combate.

Precondiciones:
Existe una partida.

Postcondiciones:
Todos los barcos están colocados correctamente.

Escenario Principal:

1. **Jugador**: Elige gestionar un barco (ver C.U. Gestionar Barco Seleccionado).
2. **Sistema**: Comprueba si están todos los barcos colocados.
   * Los pasos 1 y 2 se repiten mientras queden barcos sin colocar.
3. **Jugador**: Decide empezar a jugar.
4. **Sistema**: Guarda la posición de los barcos y empieza el juego.

Curso alternativo:
Paso 1: Coloca todos los barcos aleatoriamente.

1. **Sistema:** Mueve todos los barcos sin colocar a posiciones aleatorias donde quepan cumpliendo las normas de separación. Si no es posible colocar un barco lo omite.

Paso 1: Recoger todos los barcos.

1. **Sistema:** Retira los barcos del tablero y los mueve al puerto.

Paso 1: Girar el puerto.

1. **Sistema:** Gira todos los barcos no colocados.

Caso de Uso: Gestionar Barco

Actores:
Jugador

Descripción:
En este caso de uso el jugador selecciona un barco y decide qué hacer con él.
Las opciones que tiene son: colocarlo en el tablero, girarlo o sacarlo del tablero.

Precondiciones:
Existe un barco seleccionado.

Postcondiciones: -

Escenario Principal:

1. **Jugador:** Selecciona un barco y elige colocar, girar o retirar el barco seleccionado.
2. **Sistema:** Realiza la acción indicada por el jugador.

Curso Alternativo: -

Caso de Uso: Jugar

Actores:
Jugador

Descripción:
Cuando a un Jugador le toca el turno puede bombardear casillas del tablero de su oponente que no hayan sido bombardeadas. Cuando revela una casilla de agua pierde el turno. Si un jugador destruye todos los barcos de su oponente el juego termina y el sistema informa del resultado de la partida.

Precondiciones:
Es el turno del Jugador.

Postcondiciones:
El jugador ha hecho su jugada y termina su turno.

Escenario principal:

1. **Jugador:** Selecciona una casilla que no haya sido bombardeada.
   El jugador puede repetir el paso 1.
2. **Jugador:** Confirma que desea bombardear la casilla seleccionada si no está marcada .
3. **Sistema:** Revela la casilla al jugador.
   * Se repiten los pasos 1 a 3 mientras la casilla revelada no sea agua.
4. **Sistema:** Cede el turno al Oponente.

Curso alternativo:

Paso 2: Marcar/desmarcar una casilla.

1. **Jugador:** Puede marcar o desmarcar la casilla seleccionada indicando que no va a poder ser

bombardeada.

Paso 3: El usuario ha hundido un barco.

2. **Sistema:** Revela adicionalmente todas las casillas adyacentes al barco hundido.

Paso 3: El usuario ha hundido el último barco del Oponente.

1. **Sistema:** Informa de la victoria del Jugador sobre el Oponente y permite al jugador la opción de jugar la revancha o volver al menú principal.
2. **Jugador:** Indica si vuelve a jugar o regresa al menú principal.

Paso 4: Partida contra IA "muy fácil".

1. **Sistema:** Bombardea aleatoriamente una casilla del jugador que no haya sido bombardeada.
   * Repite hasta que el resultado sea agua

Paso 4: Partida contra IA "normal".

2. **Sistema:** Bombardea aleatoriamente una casilla del jugador que no haya sido bombardeada, preferiblemente una adyacente a una casilla de barco tocado, si existen más de una casilla del barco seguirá la linea de éste.
   * Repite hasta que el resultado sea agua

Paso 4: Gana la IA.

1. **Sistema:** Informa al jugador de su derrota del Jugador.

6.5. Modelo de dominio

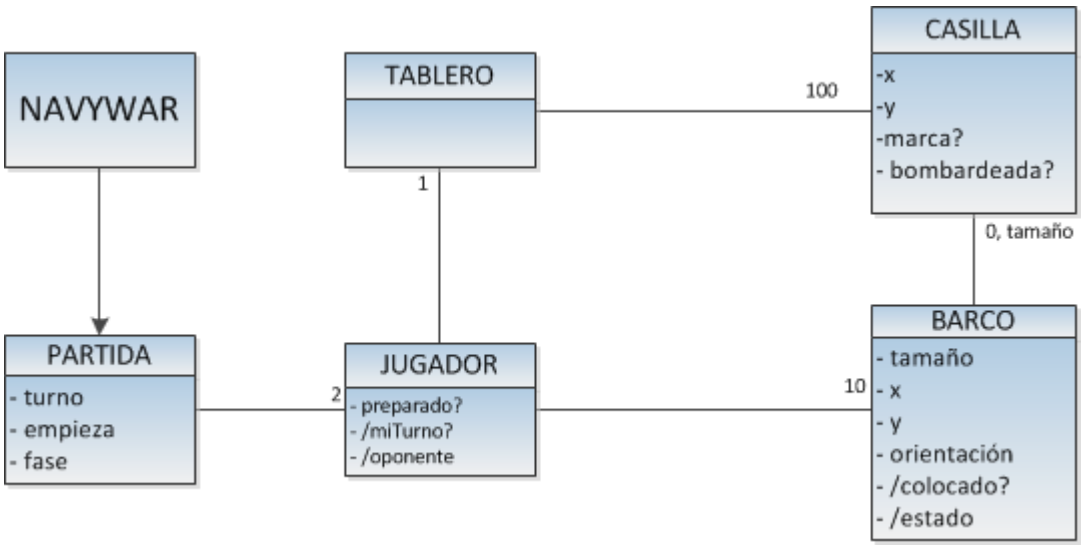


Ilustración 10 - Modelo de dominio

La ilustración 10 muestra la estructura de datos de una partida. La partida dispone de 2 jugadores aunque uno de ellos podría ser una IA, dependiendo del modo de la partida. Además la partida deberá almacenar quién empieza y en que turno se encuentra cuando alcance la fase de combate. La fase comienza en la de preparación y pasará a combate cuando ambos jugadores estén preparados.

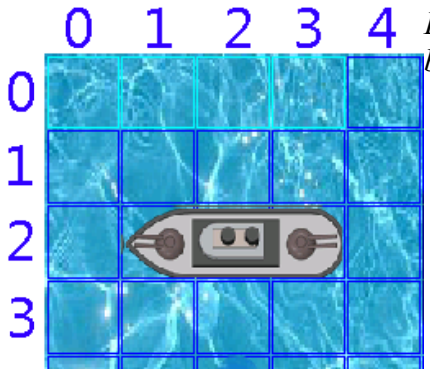
La entidad jugador no dispone de datos propios y cada jugador dispone de un único tablero. Desde un punto de vista puramente abstracto podría unirse con la entidad tablero, pero como utilizamos un diseño orientado a objetos lo mantenemos separado. Los atributos que vemos en jugador, excepto el

que indica si está preparado, son calculados:

- preparado?: Indica si el jugador está dispuesto a pasar a la fase de combate. Aunque no es calculado depende de que todos los barcos del jugador estén colocados.
- miTurno?: Indica si es el turno de dicho jugador y se calcula a partir de la fase, el jugador inicial y el turno de la partida.
- oponente?: El otro jugador de esta partida.

El tablero no contiene datos propios, pero cada uno contiene 100 casillas y 10 barcos.

Las 100 casillas de un tablero disponen una matriz de 10x10 y tienen como atributo las coordenadas. Cada casilla tiene la posibilidad de estar conectada con un barco, sólo si éste contiene dicha casilla.



En la ilustración de este ejemplo podemos ver un barco con los siguientes datos:

- *x = 1*
- *y = 2*
- *tamaño = 3*
- *orientación = HORIZONTAL*

Los 10 barcos de tablero tienen diferente tamaño de acuerdo a las reglas. Cuando colocas el barco en el tablero se enlaza a una serie de casillas limitada por su tamaño. Para almacenar su posición guardamos la esquina superior izquierda y la orientación (vertical u horizontal).

7. ANÁLISIS

En este apartado analizamos cada uno de los casos de uso del capítulo anterior para extraer las operaciones de las que se compone. De cada operación se dirán las condiciones que se deben cumplir al principio y al final de cada uno de ellos, así como su salida si es pertinente.

7.1. Caso de Uso: Iniciar Aplicación

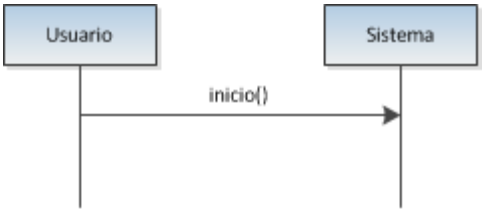


Ilustración 11 - Iniciar aplicación

7.1.1. Contrato: Inicio

Nombre:

- inicio()

Responsabilidades:

- Iniciar la aplicación.
- Cargar los datos necesarios.
- Mostrar el menú de inicio.

Precondiciones:-

Postcondiciones:

- Aplicación iniciada.
- Datos de la aplicación cargados.

Salida: -

7.2. Caso de Uso: Terminar Aplicación

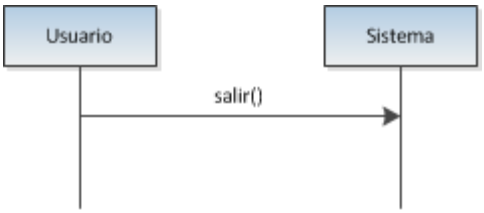


Ilustración 12 - Terminar aplicación

7.2.1. Contrato: Salir

Nombre:
- salir()

Responsabilidades:
- Terminar la Aplicación.

Precondiciones:
- Aplicación iniciada. Autorización aceptada por el usuario.

Postcondiciones:
- Aplicación terminada.

Salida: -

7.3. Caso de Uso: Crear Partida

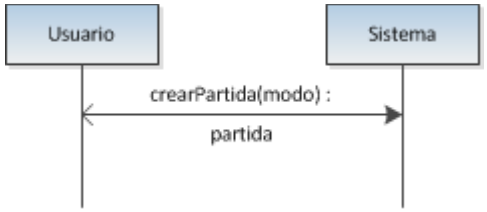


Ilustración 13 - Crear partida

7.3.1. Contrato: Crear Partida

Nombre:
- crearPartida(*modo*):*partida*

Responsabilidades:
- Crear una nueva partida con dos jugadores, uno de ellos es controlado usuario, el otro es controlado por una inteligencia artificial u otro usuario dependiendo del modo.
- Cada jugador participa en la partida con un tablero de 100 casillas y 10 barcos de acuerdo a las reglas.

Precondiciones:
- El *modo* es válido. Valores: J1F (1 jugador – IA "muy fácil"), J1N (1 jugador – IA "normal"), J2 (2 jugadores).

Postcondiciones:
- Existe una nueva partida con los 2 jugadores, sus tableros, casillas y barcos.
- Se ha iniciado la fase de preparación en dicha partida.

Salida:
- *partida*: La partida recién creada

7.4. Caso de Uso: Gestionar Flota

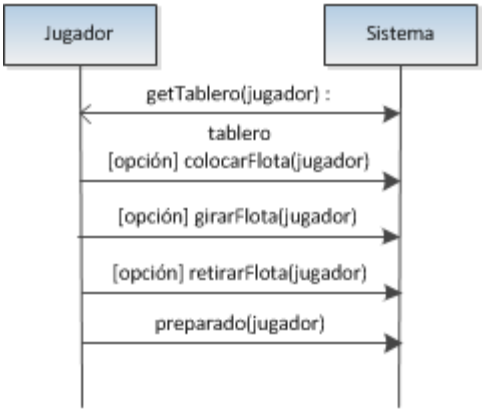


Ilustración 14 - Gestionar flota

7.4.1. Contrato: Obtener Tablero

Nombre:

- getTablero(*jugador*):tablero

Responsabilidades:

- Obtener el tablero del jugador.

Precondiciones:

- El jugador es válido y pertenece a la partida activa.

Postcondiciones: -

Salida:

- tablero: El tablero del jugador.

7.4.2. Contrato: Colocar Flota

Nombre:

- colocarFlota(*jugador*)

Responsabilidades:

- Colocar todos los barcos del jugador que no estén colocados en posiciones aleatorias.

Precondiciones:

- El jugador es válido y pertenece a la partida activa.
- Hay barcos sin colocar.

Postcondiciones:

- El máximo posible de barcos están colocados correctamente.

Salida:

7.4.3. Contrato: Girar Flota

Nombre:
- girarFlota(*jugador*)

Responsabilidades:
- Girar todos los barcos del jugador que no estén colocados en el tablero.

Precondiciones:
- El jugador es válido y pertenece a la partida activa.
- Partida en fase de preparación.
- Existen barcos sin colocar.

Postcondiciones:
- Los barcos no colocados han cambiado de orientación.

Salida: -

7.4.4. Contrato: Retirar Flota

Nombre:
- retirarFlota(*jugador*)

Responsabilidades:
- Retira todos los barcos del tablero.

Precondiciones:
- El jugador es válido y pertenece a la partida activa.
- Partida en fase de preparación.
- Existen barcos colocados.

Postcondiciones:
- No existe ningún barco colocado.

Salida: -

7.4.5. Contrato: Preparado

Nombre:
- preparado(*jugador*)

Responsabilidades:
- Cambiar el estado del jugador a preparado.
- Si los dos jugadores se encuentran preparados la partida pasa a la fase de combate.

Precondiciones:
- El jugador es válido y pertenece a la partida activa.
- Partida en fase de preparación.
- Todos los barcos están colocados.

Postcondiciones:

- La partida cambia a la fase de combate.

Salida: -

7.5. Caso de Uso: Gestionar Barco

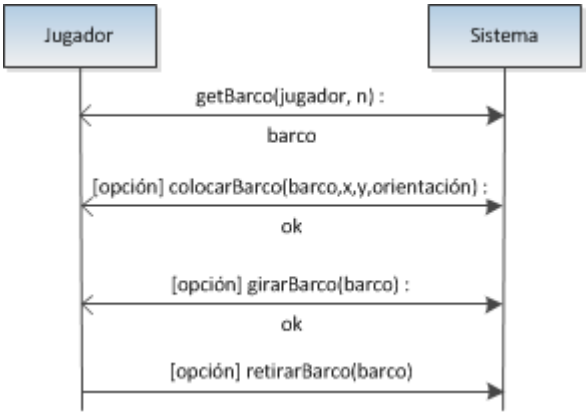


Ilustración 15 - Gestionar barco

7.5.1. Contrato: Obtener Barco

Nombre:

- getBarco(*jugador*, *n*):*barco*

Responsabilidades:

- Devuelve el barco del jugador indicado por el número *n*.

Precondiciones:

- El *jugador* es válido y pertenece a la activa.
- El índice *n* es un entero válido: $0 \leq n < 10$

Postcondiciones: -

Salida:

- barco: El barco indicado por el número n:
  - 0: portaaviones
  - 1~2: destructores
  - 3~5: fragatas
  - 6~9: submarinos

7.5.2. Contrato: Colocar Barco

Nombre:

- colocarBarco(*barco*, *x*, *y*, *orientación*):*posible*

Responsabilidades:

- Comprobar que la posición (x, y, orientación) es legal:
  - Todas las casillas que va a ocupar el barco pertenecen al tablero y no contienen barco.
  - Existe un margen de 1 casilla alrededor del barco en la nueva posición libre de barcos.
- Colocar el barco seleccionado en la posición legal indicada.

Precondiciones:

- El barco seleccionado es válido y pertenece a un jugador de la partida activa.
- Partida en fase de preparación.

Postcondiciones:

- El barco seleccionado esta colocado correctamente en la posición indicada respetando las normas de colocación.

Salida:

- posible: Si es posible colocarlo respetando las normas de colocación.

7.5.3. Contrato: Girar Barco

Nombre:

- girarBarco(*barco*):*posible*

Responsabilidades:

- Comprobar que la nueva posición es legal:
  - Todas las casillas que va a ocupar el barco pertenecen al tablero y no contienen barco.
  - Existe un margen de 1 casilla alrededor del barco en la nueva posición libre de barcos.
- Cambiar, si es legal, la orientación del barco.

Precondiciones:

- El barco seleccionado es válido, pertenece a un jugador de la partida activa y está colocado.
- Partida en fase de preparación.

Postcondiciones:

- El barco seleccionado ha alternado su orientación, respetando las normas de colocación.

Salida:

- posible: Si es posible colocarlo respetando las normas de colocación.

7.5.4. Contrato: Retirar Barco

Nombre:

- retirarBarco(*barco*)

Responsabilidades:

- Retira el barco seleccionado del tablero.

Precondiciones:

- El barco seleccionado es válido, pertenece a un jugador de la partida activa y está colocado.

- Partida en fase de preparación.

Postcondiciones:

- El barco seleccionado no esta colocado.

Salida: -

7.6. Caso de Uso: Jugar

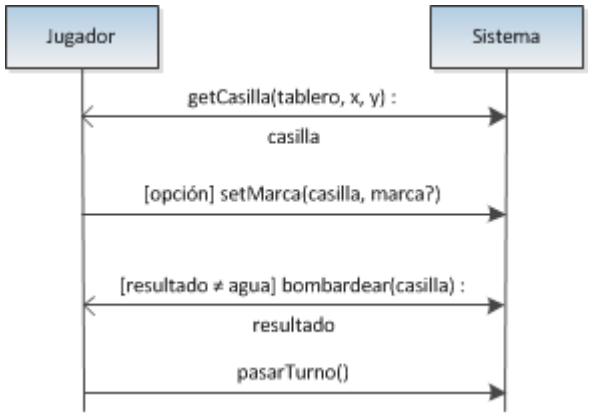


Ilustración 16 - Jugar

7.6.1. Contrato: Obtener Casilla

Nombre:

- getCasilla(*tablero*, *x*, *y*):*casilla*

Responsabilidades:

- Devuelve la casilla seleccionada.

Precondiciones:

- El *tablero* es válido y pertenece a la partida activa.
- Las coordenadas (*x*, *y*) pertenecen al *tablero*.

Postcondiciones: -

Salida:

- *casilla:* La casilla seleccionada.

7.6.2. Contrato: Marcar/Desmarcar

Nombre:

- setMarca(*casilla, marca?*)

Responsabilidades:

- Establecer si la casilla está marcada o no.

Precondiciones:

- *casilla* es una casilla existente y no bombardeada.
- Partida en fase de combate.
- El valor de *marca?* es Sí o No.

Postcondiciones:

- La casilla está bombardeada.

Salida: -

7.6.3. Contrato: Bombardear

Nombre:

- bombardear(*casilla*):*resultado*

Responsabilidades:

- Comprobar el turno del jugador.
- Si es su turno bombardear la casilla e informar del resultado.

Precondiciones:

- *casilla* es una casilla existente, no bombardeada y pertenece al jugador con el turno.
- Partida en fase de combate.
- Este turno aún no se ha tenido un resultado de Agua al bombardear.

Postcondiciones:

- Si es el turno del jugador, la casilla pasa a estar bombardeada.
- Si se bombardea la última casilla de un barco, se revelan todas las adyacentes a éste.

Salida:

- *resultado*:
  - Agua: Si no tiene barco asociado.
  - Tocado: Si el barco asociado aún tiene casillas por bombardear.
  - Hundido: Si el barco asociado no tiene casillas por bombardear.
  - Victoria: Si el jugador tiene todos los barcos hundidos.

7.6.4. Contrato: Pasar Turno

Nombre:

- pasarTurno(*jugador*)

Responsabilidades:

- Ceder el turno al oponente del jugador actual.

Precondiciones:

- Partida en fase de combate.
- Es el turno del jugador.
- En este turno el último resultado de bombardear ha sido Agua.

Postcondiciones:

- Incrementar el turno de la partida.
- Retirar el turno al jugador activo.
- Darle el turno al oponente.
- Informar al oponente de que es su turno.

Salida: -

8. DISEÑO

8.1. Contrato: Inicio

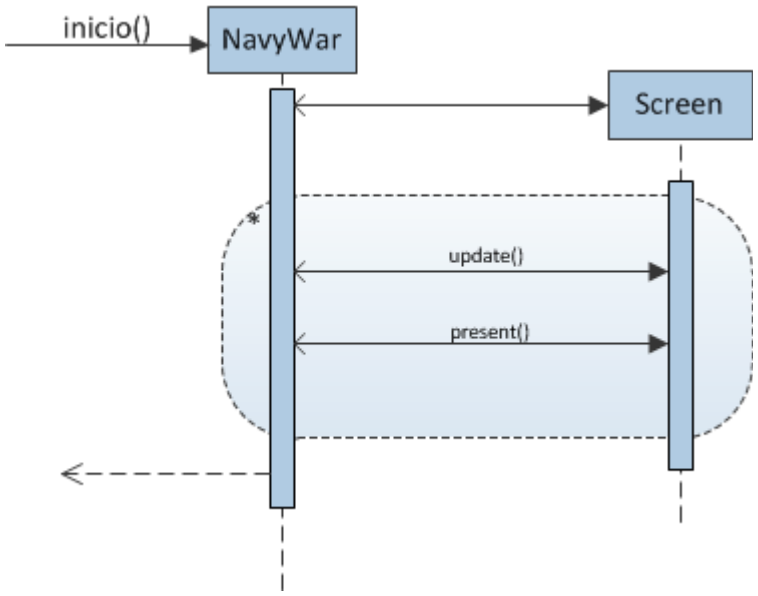


Ilustración 17 - Inicio

Primero se construye el objeto principal de la aplicación NavyWar. Éste a su vez construye un objeto Screen.

Para continuar se inicia un bucle que se repetirá un número indeterminado de veces hasta que se termine la aplicación. En dicho bucle NavyWar llama primeramente a la operación update() y luego a present() del Screen activo.

Cada Screen tiene diferentes implementaciones de sus operaciones, pero normalmente suele llamar de forma recursiva a dichas operaciones de los actores que contiene. Los casos particulares se especifican en el capítulo de Implementación de "PFC: NavyWar – Capa de Presentación" de Adrián Gil.

Aunque el bucle se detiene de forma externa utilizando la operación salir() las responsabilidades de inicio() se han cumplido y la operación finaliza sin devolver ningún resultado.

8.2. Contrato: Salir

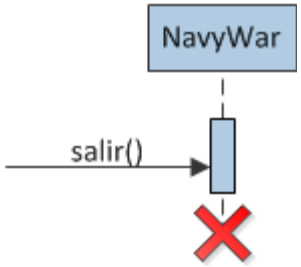


Ilustración 18 - Salir

La responsabilidad de esta operación es simplemente destruir el objeto maestro del sistema y terminar la ejecución de la aplicación. Por supuesto hay que tener en cuenta que si se destruye este objeto implica la destrucción del resto de objetos.

8.3. Contrato: Crear Partida

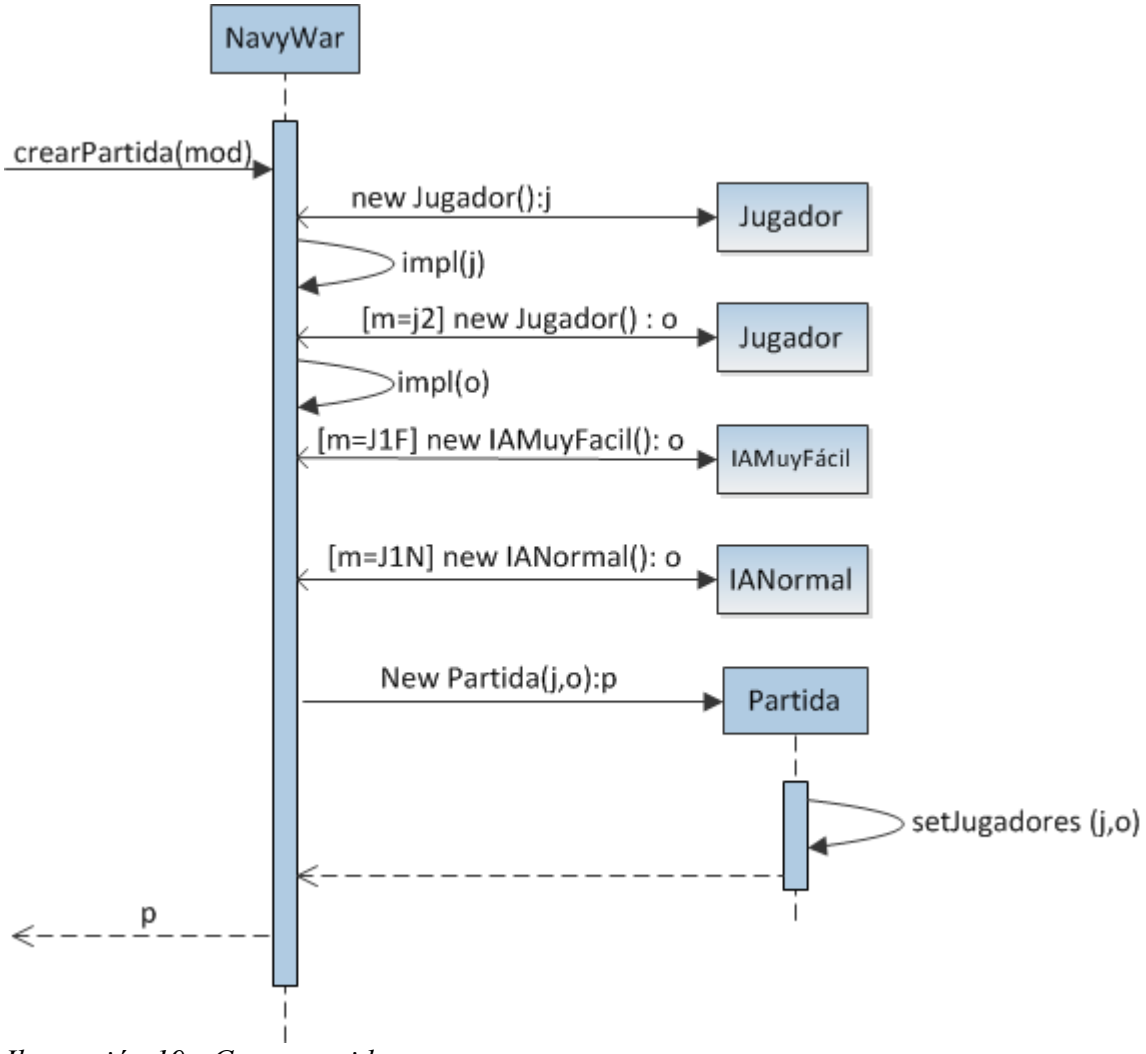


Ilustración 19 - Crear partida

NavyWar debe crear una partida con dos jugadores. Cuando se crea un jugador se inicializan todos sus datos incluyendo los demás objetos: 10 barcos sin colocar y un tablero con 100 casillas.

El primer jugador creado (*j*) es controlado por el usuario y por lo tanto debe ser implementado como una interfaz.

El segundo (*o*) puede ser controlado por el usuario o por el sistema dependiendo del modo. En el modo para dos jugadores se creará e implementará al igual que el primero.

Si el modo es contra IA "Muy fácil" se creará un objeto de la clase IAMuyFácil que extiende e implementa la clase Jugador.

En el último caso el objeto creado será de la clase IANormal que también extiende a Jugador y lo

implementa de forma ligeramente diferente a la otra IA.

Y en último lugar se crea la partida a la que se le asocian los jugadores *j* y *o*.

Finalmente la operación devuelve el objeto partida ya inicializado.

8.4. Contrato: Obtener Tablero

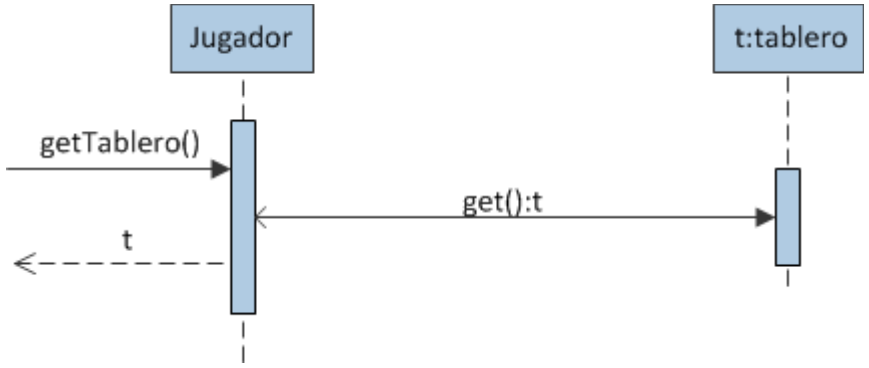


Ilustración 20 -Obtener tablero

La clase Jugador contiene un tablero. Para obtener el tablero se accede desde jugador y se devuelve su tablero asociado.

8.5. Contrato: Colocar Flota

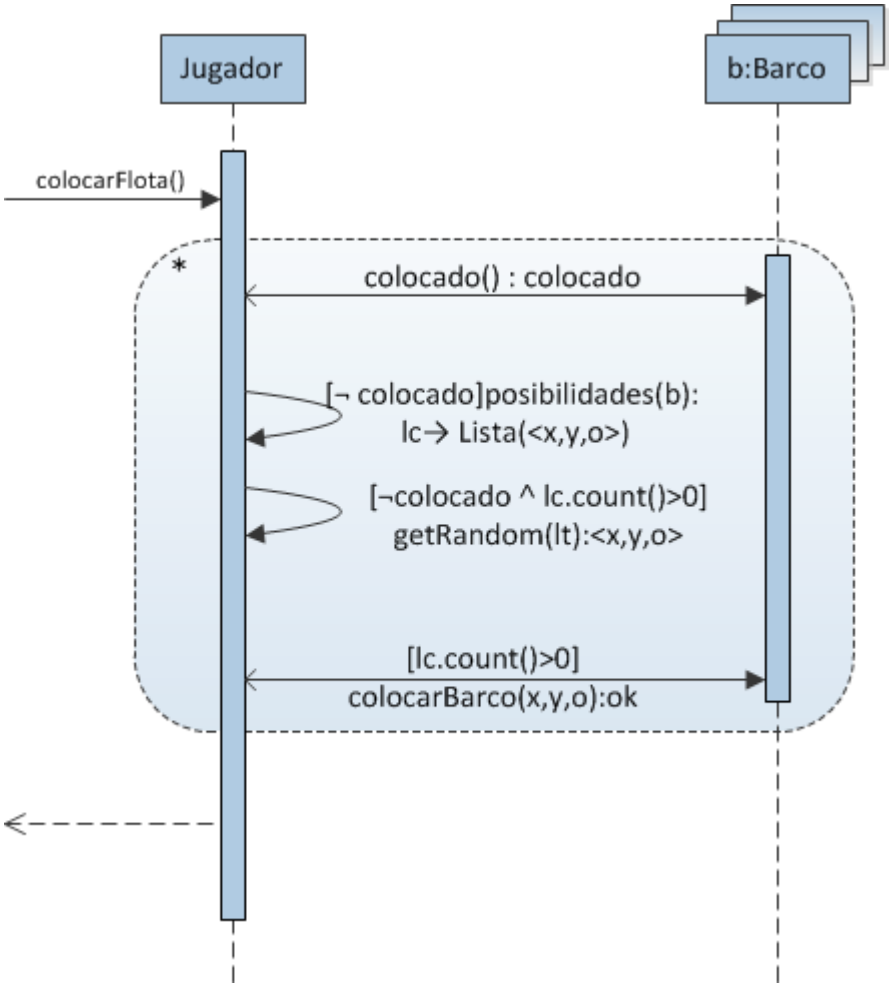


Ilustración 21 - Colocar flota

El objeto jugador que recibe la llamada a la operación colocarFlota() y recorre los 10 barcos que contiene. Por cada uno comprueba si está colocado. Si no lo estuviera calcula las posibles posiciones en las que colocarlo. Cada posición es representada por una abscisa (x), una ordenada (y) y la *orientación* (vertical u horizontal).

De la lista de posibilidades se extrae una si es posible y se coloca el barco en dicha posición. De acuerdo con las responsabilidades de esta operación, si un barco no puede ser colocado, será omitido y se continuará con el proceso hasta completarlo con los 10 barcos.

Tras terminar el bucle, la operación termina sin devolver nada.

8.6. Contrato: Girar Flota

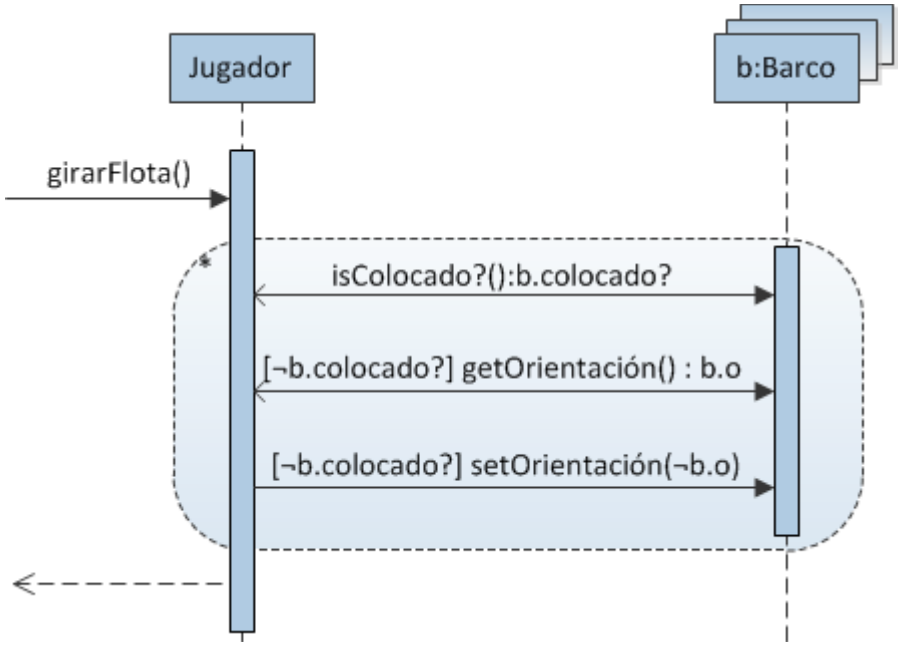


Ilustración 22 - Girar flota.

Según el Modelo de Dominio un jugador tiene 10 barcos. Esta operación recorre los 10 barcos comprobando si están colocados. En caso de no estarlo se cambia su orientación.

Así cumplida la responsabilidad del contrato termina la operación sin devolver nada.

8.7. Contrato: Retirar Flota

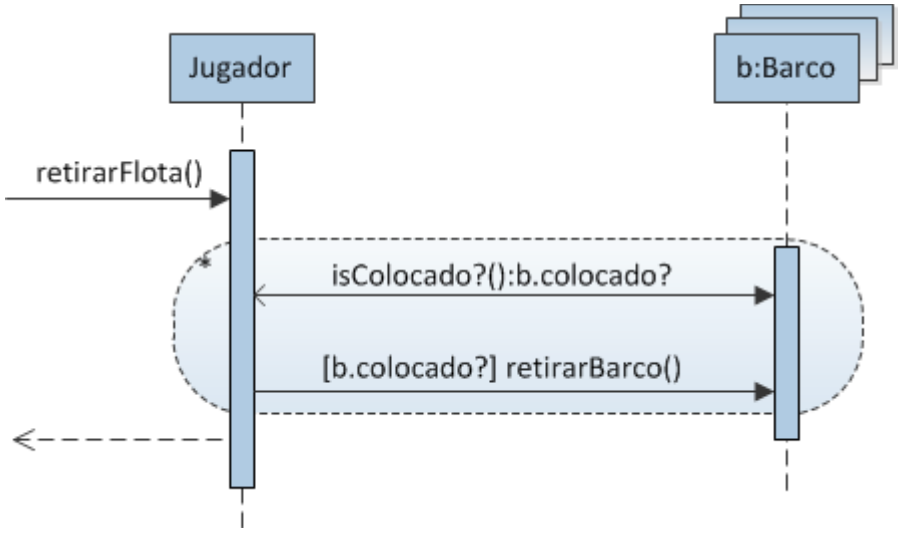


Ilustración 23 - Retirar flota

Cómo hemos indicado antes, según el MD un jugador tiene 10 barcos. El contrato requiere retirar cada barco que se encuentre colocado, de modo que se recorren los 10 barcos comprobando si están colocados. Si un barco está colocado se retira mediante la operación retirarBarco(barco).

Una vez completado el proceso por cada barco la operación termina sin devolver ninguna salida.

8.8. Contrato: Preparado

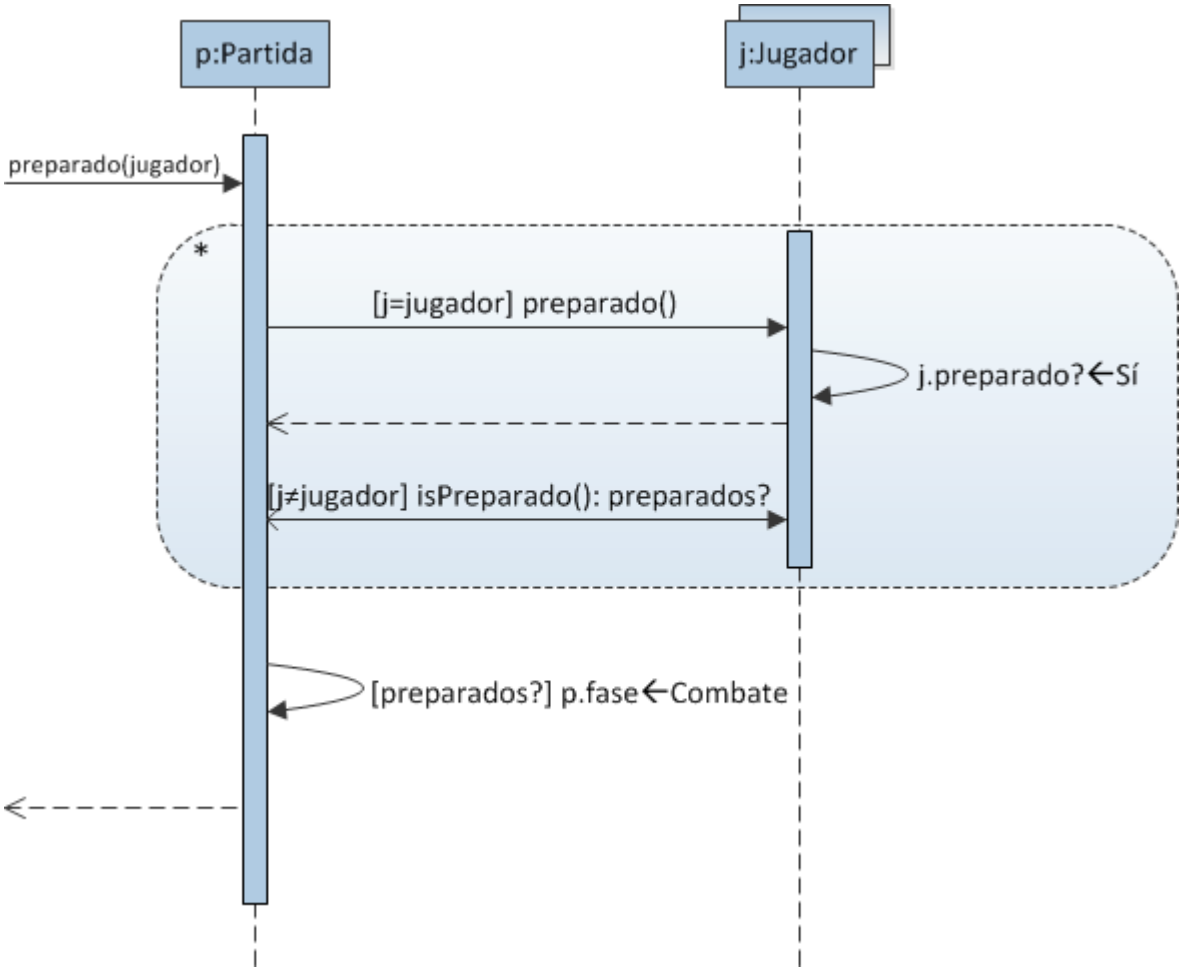


Ilustración 24 - Preparado

La operación preparado(jugador) se efectúa sobre la partida activa en lugar de directamente sobre el jugador porque afecta a ambos.

La operación recorre los dos jugadores comprobando si se trata del jugador activo, indicado por el parámetro de la operación. En este caso se le indica al jugador que cambie su estado a preparado. En caso contrario se comprueba si el oponente del jugador activo se encuentra ya preparado.

Al terminar de recorrer los dos jugadores si ambos están preparados se cambia la fase de la partida a "Combate". Y finalmente, termina la operación sin devolver ninguna salida.

8.9. Contrato: Obtener Barco

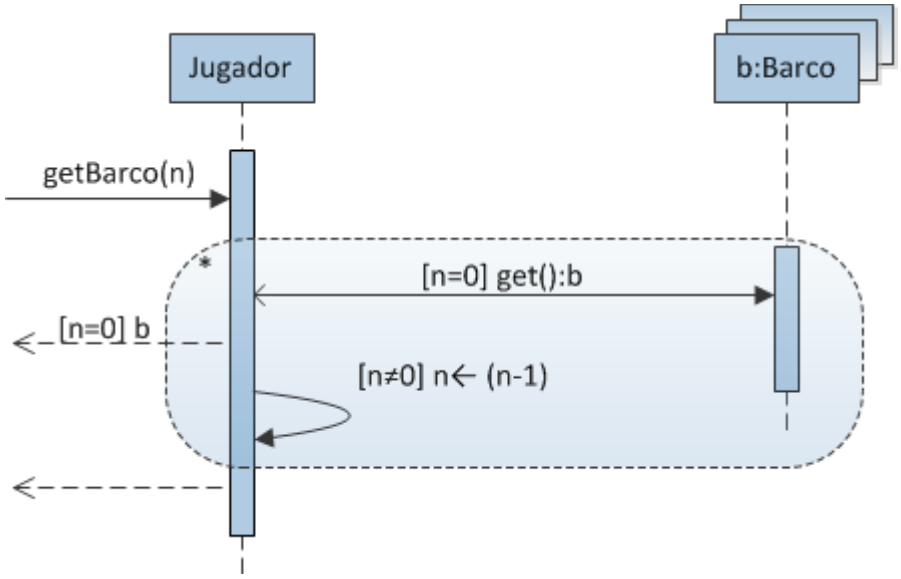


Ilustración 25 - Obtener barco

Según el MD un jugador contiene10 barcos, que numerados del 0 al 9 son:

- 0: El portaaviones.
- 1 y 2: Los dos destructores.
- 3 a 5: Las tres fragatas.
- 6 a 9: Los cuatro submarinos

La operación getBarco(jugador, n) tiene un parámetro *n*, que indica ese número. Se iterará por todos los barcos del jugador, reduciendo el índice en 1 cada vez cuando *n* sea 0 significa que el barco es el indicado y se terminará la operación devolviendo dicho barco sin esperar a completar todas las iteraciones.

Si se terminan las 10 iteraciones y no se ha devuelto ningún barco significa que no se cumplen las precondiciones de la operación y se terminará la operación sin devolver nada.

8.10. Contrato: Colocar Barco

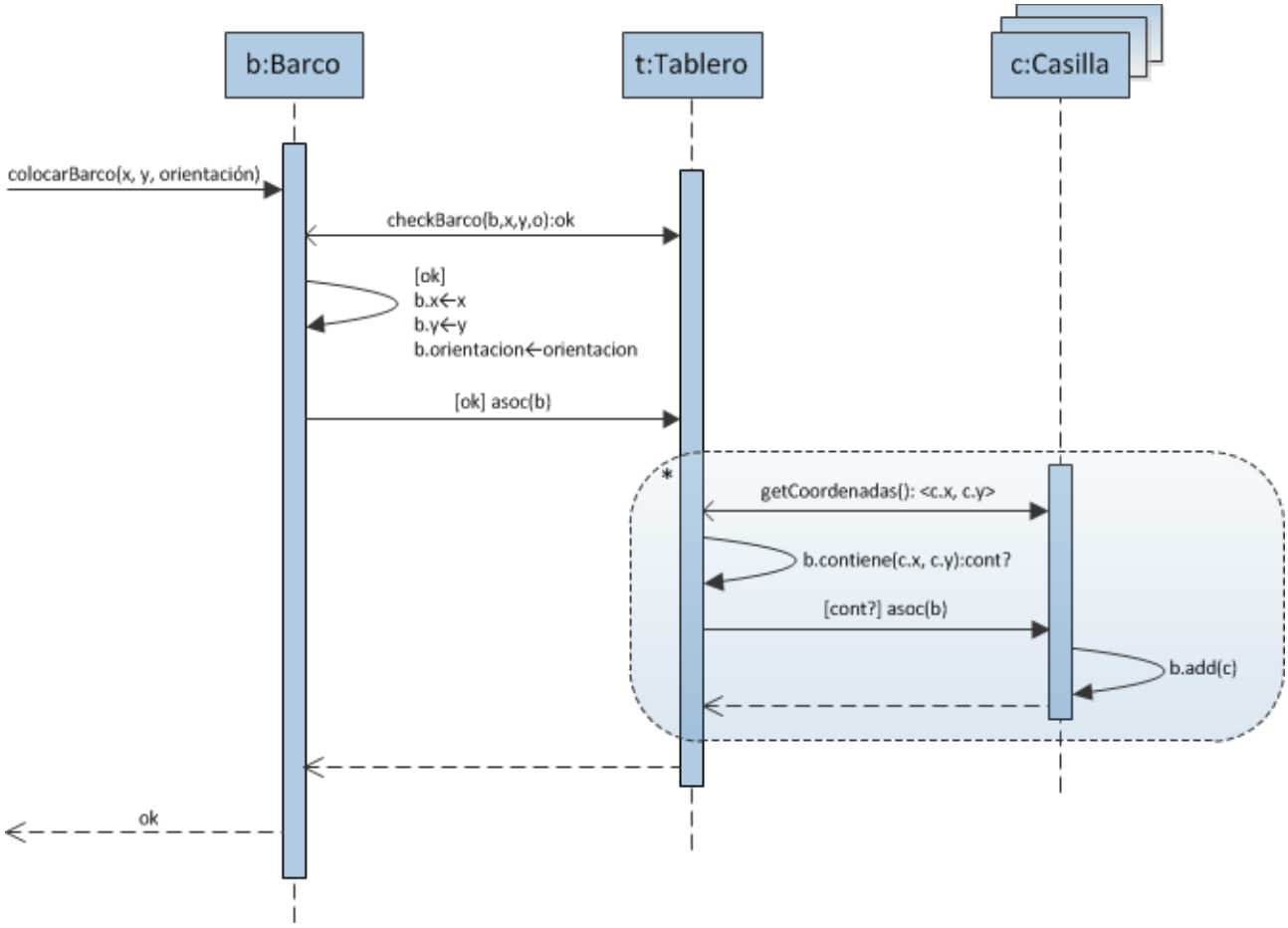


Ilustración 26 – Colocar barco

En la ilustración 26 hemos simplificado la relación entre las clases Barco y Tablero. Para facilitar la lectura hemos obviando el paso intermedio por Jugador, ya que Jugador y Tablero tienen una relación 1:1.

Primeramente hemos de comprobar que la posición indicada por la operación es válida, esto se hará mediante la operación check(barco, x, y, orientación). Esta operación examina las casillas que ocuparía el barco si se colocase en dicha posición comprobando que se encuentren dentro del tablero. También calcula un rectángulo con la nueva posición y un margen de 1 casilla en todas direcciones. Finalmente comprueba que las casillas contenidas en la intersección de dicho rectángulo con el tablero no estén asociadas a ningún barco. Al terminar la operación, devuelve un valor verdadero o falso que indica si colocar el barco en la posición indicada cumple las normas de colocación.

Si se cumplen las normas de colocación se establecen los atributos de la nueva posición del barco. Seguidamente se asocia el barco con las casillas sobre las que se está colocando. Para lograrlo el Jugador accede a su Tablero y desde éste a su vez a las casillas de las que se obtienen sus coordenadas.

Luego se comprueba que las coordenadas son las de una casilla sobre la que se está colocando el barco. En caso afirmativo se asocia el barco con la casilla y viceversa.

Una vez terminada la asociación la operación colocarBarco(barco, x, y, orientación) devuelve el resultado de la comprobación de las reglas de colocación realizada anteriormente.

8.11. Contrato: Girar Barco

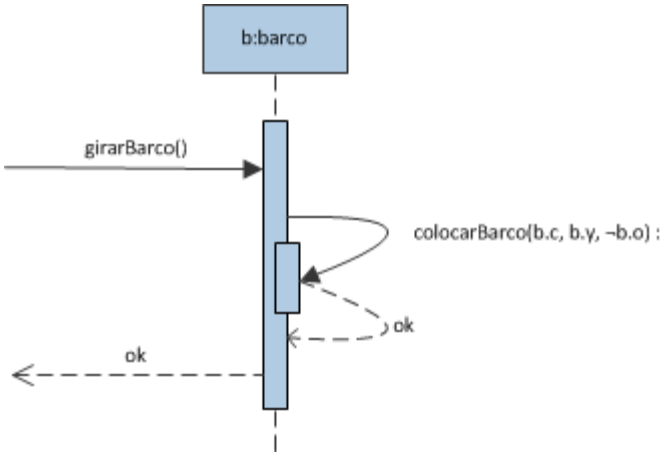


Ilustración 27 - Girar barco

El diseño de este contrato funciona de forma similar al de Colocar barco (8.10). Como se puede apreciar en la ilustración 27 se emplea la operación colocarBarco para colocar el barco en las mismas coordenadas pero alterando la orientación.

El resultado de la operación al igual que en la anterior es un valor verdadero/falso que indica si la operación ha surtido efecto o el barco permanece inalterado.

8.12. Contrato: Retirar Barco

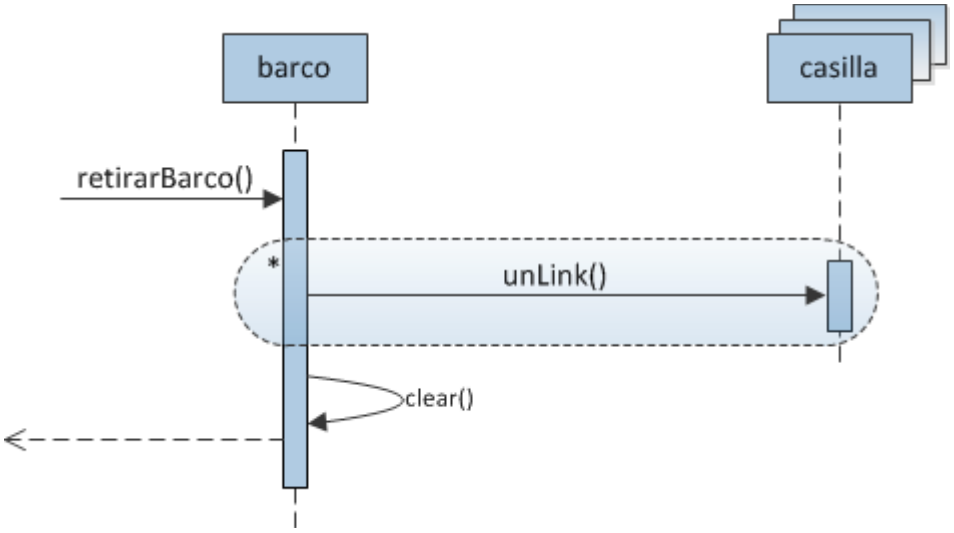


Ilustración 28 - Retirar barco

a operación retirarBarco(barco) recorre todas las casillas asociadas al barco y rompe su asociación. Tras esto se vacía la lista de casillas del barco confirmando la separación por ambos lados.

La poscondición se ha cumplido y la operación termina sin devolver nada.

8.13. Contrato: Obtener Casilla

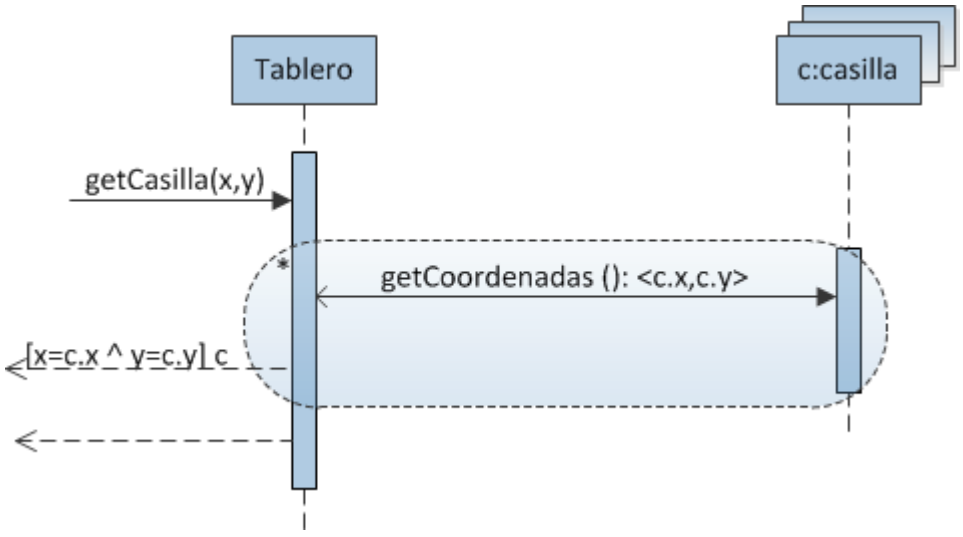


Ilustración 29 - Obtener casilla

El tablero contiene sus 100 casillas y será la clase sobre la que se efecturá la operación getCasilla(tablero, x, y).

Se recorren las 100 casillas una a una obteniendo sus coordenadas y comparándolas con las indicadas en la operación. Si coinciden se termina el bucle y se devuelve la casilla.

Si se termina de recorrer todas las casillas sin encontrar coincidencias significa que la precondición no se está cumpliendo por lo tanto no se devuelve nada y se termina la operación.

8.14. Contrato: Marcar/Desmarcar

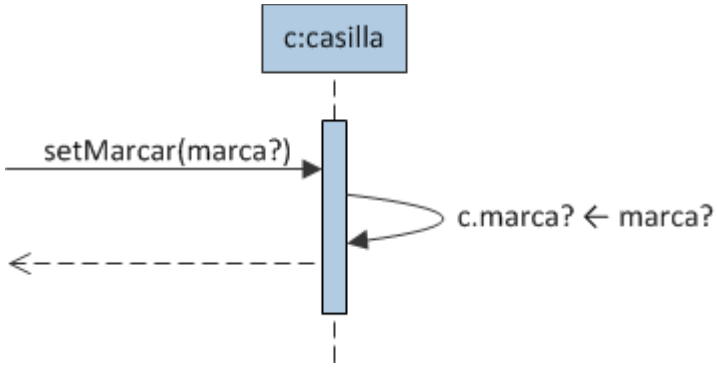


Ilustración 30 - Marcar/Desmarcar

Esta operación modifica el valor del atributo *marca?* según lo indicado por el parámetro y retorna sin devolver nada.

8.15. Contrato: Bombardear

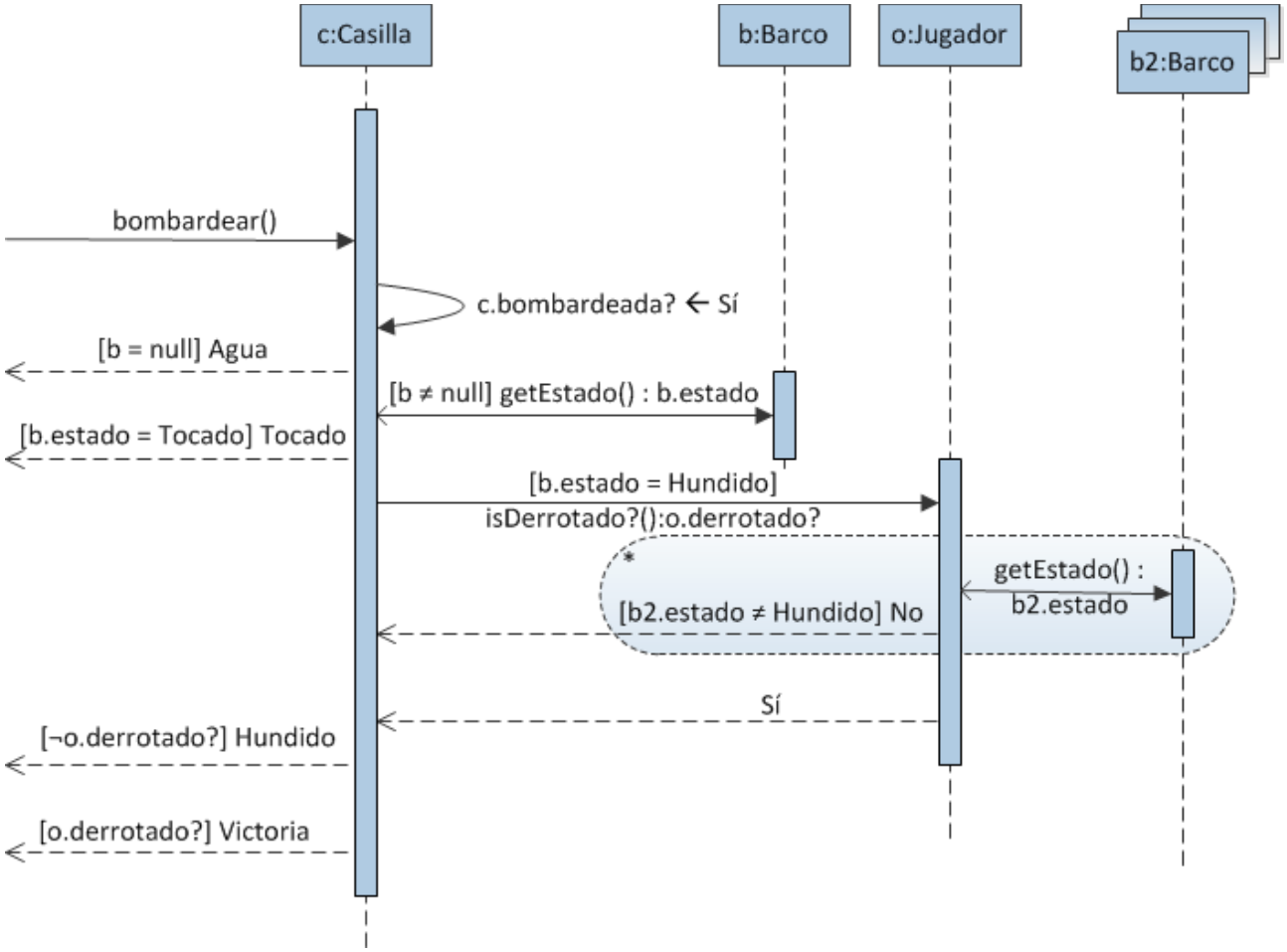


Ilustración 31 - Bombardear

En la ilustración 31 hemos simplificado la relación Casilla-Tablero-Jugador omitiendo la clase intermedia. Esto es posible debido a que Tablero y Jugador tienen una relación 1:1.

La clase que recibe la operación bombardear(casilla) es Casilla. Lo primero es el cambio del valor del atributo *bombardeada?* de la casilla a verdadero. Tras esto comprueba el si tiene un barco asociado. Si no tiene ningún barco asociado retorna el resultado "Agua".

De tener un barco comprueba su estado. El estado del barco se calcula comprobando sus casillas asociadas y puesto que una ya sabemos que ha sido bombardeada, los dos resultados posibles son: "Tocado" o "Hundido". De darse el primer caso, el retorno sería "Tocado".

Si el estado del barco fuese "Hundido" existiría la posibilidad de haber bombardeado el último. Para comprobarlo necesitamos consultar al jugador a quien pertenece la casilla. La clase Jugador comprueba si ha sido derrotado recorriendo todos sus barcos. Por cada barco comprueba su estado. Si encuentra alguno cuyo estado no sea "Hundido" la derrota es negativa. Por el contrario, si termina el bucle con

todos los barcos hundidos la derrota del jugador bombardeado es positiva.

Finalmente contando con la información de la derrota del oponente del jugador activo, la operación termina devolviendo "Hundido" si no ha sido derrotado o "Victoria" en caso contrario.

8.16. Contrato: Pasar Turno

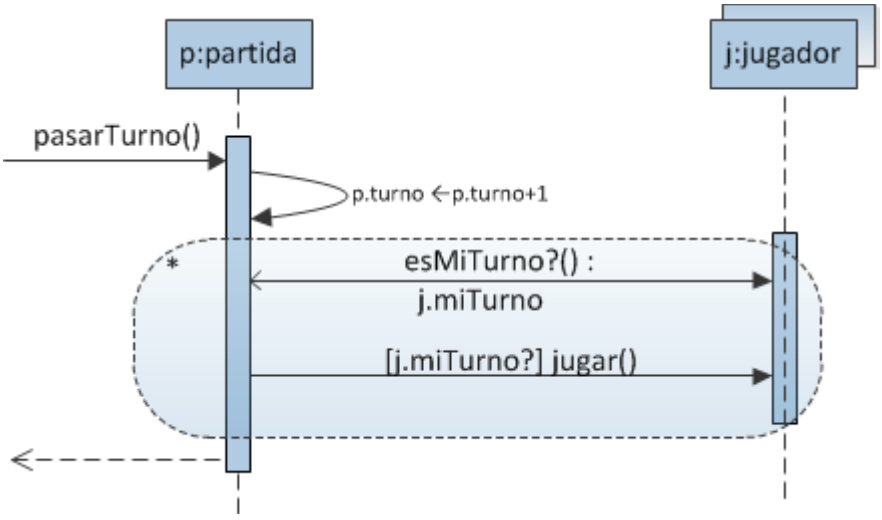


Ilustración 32 - Pasar turno

La operación pasarTurno() modifica el turno de la partida actual incrementándolo en 1.

La segunda parte de las responsabilidades implica informar al jugador que obtiene el turno. Para lograrlo accedemos a los dos jugadores de la partida y comprobamos si es el turno de ese jugador, Si se da el caso utilizamos la operación jugar() del jugador que le indica que es su turno. Esta operación es implementada por el sistema, sólo si es una IA, de otro modo es implementada por la capa de presentación. La implementación de esta operación en el caso de las IA's se especifica en el capítulo "Implementación" de la memoria "PFC: NavyWar – Capa de dominio" de Daniel Crego.

Finalmente la operación termina sin devolver ninguna salida.

9. IMPLEMENTACIÓN DE LA CAPA DE PRESENTACIÓN

Para el desarrollo de la capa de presentación se han utilizado las clases actor y actorgroup explicadas anteriormente. En este apartado explicaremos que clases exactamente se han utilizado.

Paquetes de la capa de dominio:

Paquete	Descripción
com.navywar.logic	Implementación del Modelo de Dominio
com.navywar.logic.data	Gestión de datos y permanencia
com.navywar.logic.ia	Implementación de Inteligencias Artificiales
com.navywar.namespace.ex	Paquete de excepciones

Paquetes y directorios de la capa de presentación:

Paquete	Descripción
com.navywar	Pantallas del juego
com.navywar.framework	Paquete de interfaces
com.navywar.framework.impl	Interfaces implementadas
com.navywar.graphics	Paquete de actores
com.navywar.graphics.dialogs	Clases que utilizan la librería de Android

Directorio	Descripción
assets	Recursos audiovisuales
res/anim	Definiciones XML de animaciones
res/values	Definiciones XML de constantes
res/layout	Definiciones XML de los menús
res/drawable	Recursos gráficos y sus definiciones XML

AndroidManifest.xml

En este fichero se definen las propiedades de la aplicación, como los permisos que requiere o las actividades y servicios que utiliza.

9.1. Diagrama de clases

En la siguiente imagen se muestran las clases que se han utilizado para el desarrollo de NavyWar y la relación que existe entre ellas. Como se puede ver todas ellas extienden de actor que es la clase principal de este desarrollo.

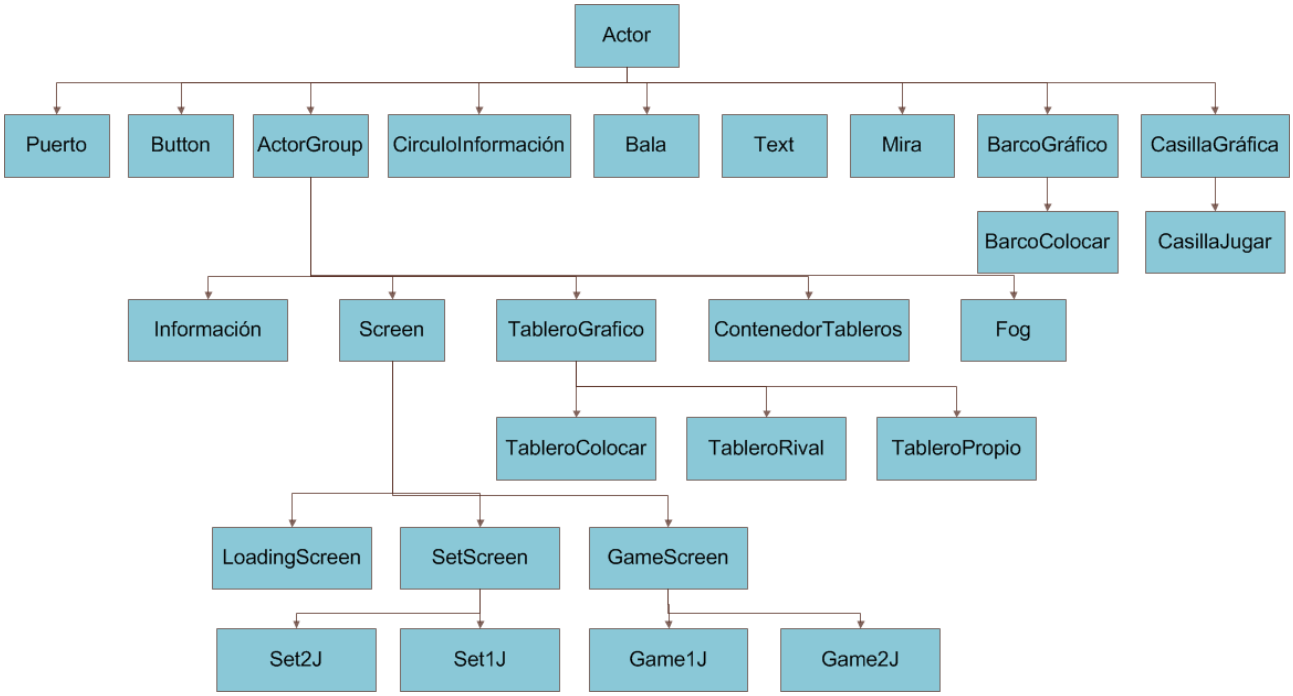


Ilustración 33: Diagrama de clases

9.2. Explicación de clases

9.2.1. com.navywar.graphics

Actor

Como ya se ha comentado es la base de todo el desarrollo gráfico. Es una clase abstracta que representa a un actor del juego. Todos los actores están dentro de un actorgroup por lo que al crear un actor debemos indicar a que actorgroup pertenece.

Tiene tres funciones abstractas para gestionar los tres toques de pantalla:

- onTouchDown: Esta función se ejecutará cuando el actor sea pulsado.

- onTouchUp: Esta ocurre cuando el usuario deja de pulsar al actor.
- onTouchDrag: Esta ultima ocurre cuando el usuario arrastra al actor.

Las tres funciones reciben un objeto Event con los datos del toque. Para comprobar si se ha recibido el toque actor cuenta con la función onBounds(x, y) que devuelve true si el punto(x,y) se encuentra dentro del actor.

Otra de sus funciones abstractas es update que se ejecuta periódicamente. Se utiliza si el actor tiene que estar continuamente realizando una acción.

También tiene funciones para darle una posición respecto a su actorgroup o respecto a la pantalla (setRelX, setRelY, setX, setY) y para darle un alto y un ancho(setWidth, setHeight).

Para definir como se muestra el actor cuenta con la función abstracta paint que pintará al actor.

Por ultimo la función setActivo nos permite anular al actor para que no reciba toques, se actualiza ni se pinta.

ActorGroup

Esta clase abstracta que extiende de actor tiene una funcionalidad distinta. Se encarga de gestionar a un grupo de actores.

Al extender de actor implementa sus funciones abstractas.

- Update: En esta función actorgroup recorre a sus actores llamando a sus funciones update.
- OnTouch: Las funciones de toque funcionan igual que en un actor. Solo las recibe un actor group que ha recibido un toque de ese tipo. Ademas en cada una de ellas el actorgroup recorre su lista de actores comprobando cual ha recibido el toque y llamando a la función correspondiente de ese actor.
- Paint: Esta función recorrerá los actores llamando a su función paint.

Actorgroup tiene 2 funciones más para pintarse:

- paintBackground: Esta función abstracta se ejecuta antes de pintar a los actores del actorgroup.
- PaintForeground: Esta función se ejecuta después de pintar a los actores.

CasillaGráfica

Este actor representa a una casilla del tablero. Tiene una casilla lógica que se le indica al crearse ademas de los campos de posición, tamaño y actorgroup.

Tiene las funciónes

- getCasilla que devuelve la casilla lógica
- setCasilla que cambia la casilla lógica.

Ademas extiende las funciones:

- update: cambia su color dependiendo del estado de la casilla lógica
- paint: pinta un recuadro del color que tenga asociado.

Casilla Juego

Esta clase extiende de casillaGráfica y representa una casilla del tablero que puede ser bombardeada.

Tiene las funciones:

- canBomb: Nos indica si la casilla puede ser bombardeada.
- marcar(): marca la casilla o la desmarca si estaba marcada.

Ademas extiende las funciones:

- update: cambia el color de la casilla y activa la animación cuando es bombardeada.
- paint: si está en un tablero rival añade un clipping para recortar la niebla.

TableroGráfico

Este actorgroup es también una clase abstracta y representa a un tablero del juego. Al crearlo hay que darle un actorgroup, una posición (x,y) un tamaño(alto,ancho) y un tablero lógico.

Ademas cuenta con las funciones:

- getCasillaGraficaPos(x,y) que nos devuelve la casilla en la posición (x,y).
- setImagen, getImagen: funciones para gestionar la imagen de fondo del tablero.
- getTablero: Esta función devuelve el tablero lógico asociado al tableroGráfico.
- cargarCasillas: En esta función abstracta se crean las 100 casillas que contiene.

Tablero Colocar

Este tablero se utiliza en la fase de preparación para colocar los barcos. En su función cargarCasillas se cargan 100 casillasGráficas.

Tablero Propio

Este tablero se utiliza en la fase de combate y representa al tablero en el que has colocado tus barcos.

En su función cargarCasillas carga 100 casillaJuego.

Implementa la función paintForegroud en la que busca las casillas que tiene un barco y han sido tocadas y las pinta encima del tablero.

Tablero Rival

Este tablero también se utiliza en la fase de combate y representa al tablero que debes bombardear.

En su función cargarCasillas carga 100 casillaJuego, un fog y una mira.

Barco Gráfico

Este actor representa un barco del juego. Al crear uno debemos darle los datos de posición(x,y) ademas de su grupo y de un barco lógico. Según el tamaño del barco lógico se le asocia la imagen correspondiente.

También cuenta con las funciones.

- girar: esta función gira el barco si es posible.
- getBarco: devuelve el barco lógico asociado.
- SetBarco: cambia el barco lógico.

Barco Colocar

Esta clase extiende de barco gráfico y se utiliza en la fase de preparación.

Implementa las funciones:

- onTouchUp: Comprueba si esta en el tablero o fuera y coloca el barco en la casilla o en el puerto o en la ultima casilla en la que ha estado colocado.
- onTouchDown: Guarda la casilla en la que esta colocado, si esta colocado.
- onTouchDrag: mueve el barco según el movimiento del dedo.

Mira

Este actor se utiliza en la fase de combate y sirve para seleccionar la casilla que se quiere bombardear.

Implementa las función onTouchUp para bombardear la casilla sobre la que se encuentra.

Bala

Esta clase representa a la bala que se utiliza en la fase de combate para simular un bombardeo a una casilla.

Puerto

El puerto se utiliza en la fase de preparación para gestionar los barcos sin colocar. Tiene asignada una posición para cada tipo de barco en vertical y en horizontal.

Tiene las funciones:

- girar(): gira todos los barcos que están en el puerto.
- atracar: a esta función se le pasa un barco gráfico y lo descoloca del tablero y lo mueve al puerto.

Button

La clase asbtracta button es un actor que representa a un botón.

Tiene la función abstracta action que se ejecuta al pulsar el botón.

Implementa la función onTouchUp que llama a actión y reproduce un sonido.

Información

Este actor se utiliza en la fase de combate para informar del estado de la flota. Pinta 1 circulo por cada barco del juego y lo cambia de color cuando se ha hundido ese barco.

Circulo Información

Se utiliza dentro de información, representa el estado de un barco. Tiene asociado un barco lógico y cambia su color según el estado de ese barco.

Text

A este actor hay que darle una posición, un ancho y un texto al crearlo. El calculará el tamaño de letra que necesita para llenar todo el ancho que se le ha indicado.

Implementa la función paint dibujando el string.

Fog

Se utiliza en el tablero rival de la fase de combate. Representa la niebla que se sobrepone al tablero.

Implementa las funciones:

- paintBackground: añade el clipping que han realizado las casillas del tablero.
- PaintForeground: se pinta y retira el clipping.

Ademas este actorgroup contiene 4 actores fog que se mueven para simular el efecto de la niebla.

9.2.2. com.navywar.framework

En esta carpeta se encuentran las interfaces del juego. Pero las que más nos interesan son la clase Screen y la clase Game.

Screen

Este actorgroup es de los más importantes de la capa de presentación. Representa una pantalla del juego. Screen, pese a ser un actor, no tiene un actorgroup asociado ya que es la clase que inicia el bucle de actualizar pintar. Al crear un screen pide un game.

Implementa las funciones:

- paintBackground: En esta función pinta el fondo de la pantalla.
- update: Recoge la lista de eventos producidos y llama la función onTouch de actorgroup correspondiente para cada tipo de evento. De esa forma la función del actorgroup recorrerá la lista de actores para comprobar sobre cual se ha producido el evento.

Ademas tiene las funciones abstractas:

- pause: Se ejecuta si el juego se pausa.
- resume: Se ejecuta al reanudar el juego.
- dispose: Se ejecuta al cerrar el juego.
- present: Recorre su lista de actores y les indica que se pinten.

Game

Esta interfaz representa al juego entero,

Tiene las funciones:

- getInput: devuelve el controlador de eventos.
- getGraphics: devuelve el controlador de gráficos.
- getAudio: devuelve el controlador de sonido.
- getPartida: devuelve la partida actual.
- getCurrentScreen: devuelve el screen actual.
- setCurrenteScreen: esta función pide una screen y reemplaza la screen actual.

9.2.3. com.navywar

En esta carpeta se encuentran todas las pantallas del juego.

Loading Screen

En esta pantalla se realiza la carga de imágenes. La imágenes se guardan en un objeto Assets al que se accede más adelante.

NavySetScreen

Esta pantalla se encarga de la fase de preparación. Al crearla hay que darle un Game y un jugador. El jugador sera del que se saquen en tablero, los barcos y las casillas lógicas. En esta pantalla se utilizan 1 tablero, 1 puerto y 10 barcos. El button de colocar o pasar de fase es abstracto.

Set1J y Set2J

Estas dos clases extienden de NavySetScreen y cambian la función del botón colocar/pasar.

GameScreen

Esta pantalla se encarga de la fase de combate. Al crearla hay que darle un Game y un jugador. Tiene dos funciones abstractas, cargarTableros y actionBotonPasar que se encarga de la carga de tableros y de la acción del botón de pasar de turno.

Game1J y Game2J

Estas dos clases implementan las funciones abstractas de GameScreen para cada tipo de juego.

9.2.4. Assets

En este directorio se encuentran los recursos gráficos y de sonido que se utilizan durante el juego.

9.2.5. Res/layout

En este directorio podemos encontrar los layouts utilizados en la aplicación.

En Android, cada pantalla de una aplicación se carga desde un fichero XML que actua de recurso. Un layout es un recurso que se usa en Android para componer las pantallas de nuestra aplicación.

En un layout podemos añadir recursos de Android como botones o listas.

9.2.6. Res/values

Aquí se encuentran los datos estáticos utilizados por la aplicación, como textos o tamaños. Estos archivos nos permiten adaptar la aplicación a otros tamaños o idiomas generando nuevos ficheros con los datos adaptados.

9.2.7. Res/anim

Para generar movimientos y animaciones Android tiene definidas unas propiedades que se definen en ficheros xml. En este directorio se encuentran esas animaciones que se asignan a los recursos del layout en sus propiedades.

9.2.8. Res/drawable

Ademas de imágenes en Android se pueden generar gráficos mediante archivos xml. En estos archivos definimos que propiedades se le asignaran al recurso correspondiente.

10. GESTIÓN

Este proyecto se planificó para ser realizado en unos 3 meses, desde septiembre de 2012 hasta enero dejando un mes antes de la entrega para poder realizar la memoria. Sin embargo a los dos meses del comienzo del proyecto ya nos dimos cuenta de que cumplir con esa fecha iba a ser muy complicado. Al estar realizado por dos personas muchos aspectos del desarrollo mejoran, pero la gestión cambia, y esto es algo que no habíamos previsto.Debido a esto nos vimos forzados a alargar el proyecto hasta junio de 2013. Los motivos serán detallados a continuación.

El inicio del proyecto fue lento y complicado ya que ninguno de los miembros había trabajado antes sobre la plataforma Android y la mayor parte del tiempo fue invertido en aprender cómo funcionaba el sistema operativo y qué herramientas ofrecía. Este periodo duró aproximadamente un mes y en él se realizaron pequeñas pruebas de funcionamiento del sistema para aprender y apenas se consiguió avanzar.

Tras esta fase de aprendizaje el proyecto avanzó considerablemente. Desde inicios de octubre a mediados de noviembre los integrantes consiguieron montar las bases lógicas y gráficas del sistema y montar dos versiones del juego que cumplían con múltiples de los objetivos planteados. El primer prototipo se consideró completada a finales de octubre. No contaba con mucha calidad gráfica pero sí que cumplía las reglas del juego a la perfección y permitía jugar contra la máquina de una manera bastante cómoda.

La segunda versión se declaró completada en apenas dos semanas de trabajo. Se mejoraron todos los aspectos gráficos del juego incluyendo animaciones y imágenes. se montaron nuevas dificultades de juego y un menú que permitía seleccionarlas.

Inspirados por los logros anteriores nos lanzamos a realizar una versión de juego on-line que fue lo que provocó que la entrega del proyecto se aplazara. Durante un mes el avance volvió a ser muy lento debido de nuevo al desconocimiento de la tecnología y lo único que se logró fue montar una base de datos en un servidor y agregar nuevos jugadores desde el dispositivo móvil.

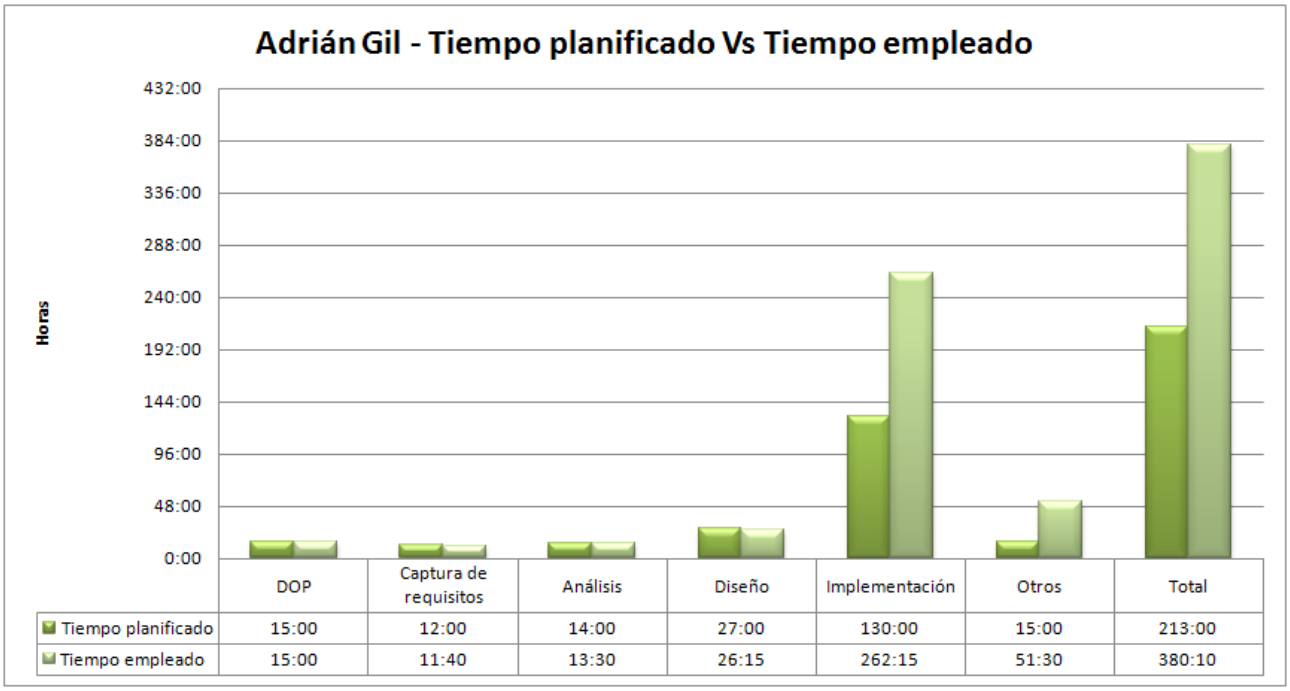
Durante navidad el avance se centró en realizar la memoria del proyecto, pero las festividades dificultaron su realización y ésta se completo a principios de febrero.

Como la entrega planificada ya era inviable decidimos mejorar el proyecto en lo posible. Sin embargo esta tarea resultó más difícil de lo que parecía por lo que ambos integrantes participaron activamente en las dos partes del proyecto.

En la parte gráfica la mejora se centró en incluir pantallas, menús y otras funcionalidades que ofrece Android. Este objetivo requirió de mucho aprendizaje y multitud de pruebas. Como resultado de esta investigación se consiguió incluir un menú de opciones que aparece al pulsar el botón inferior, un formulario para registrar al usuario en la parte on-line y para que este pueda iniciar sesión y una lista de partidas con la cualidad de actualizar si esta es arrastrada hacia abajo.

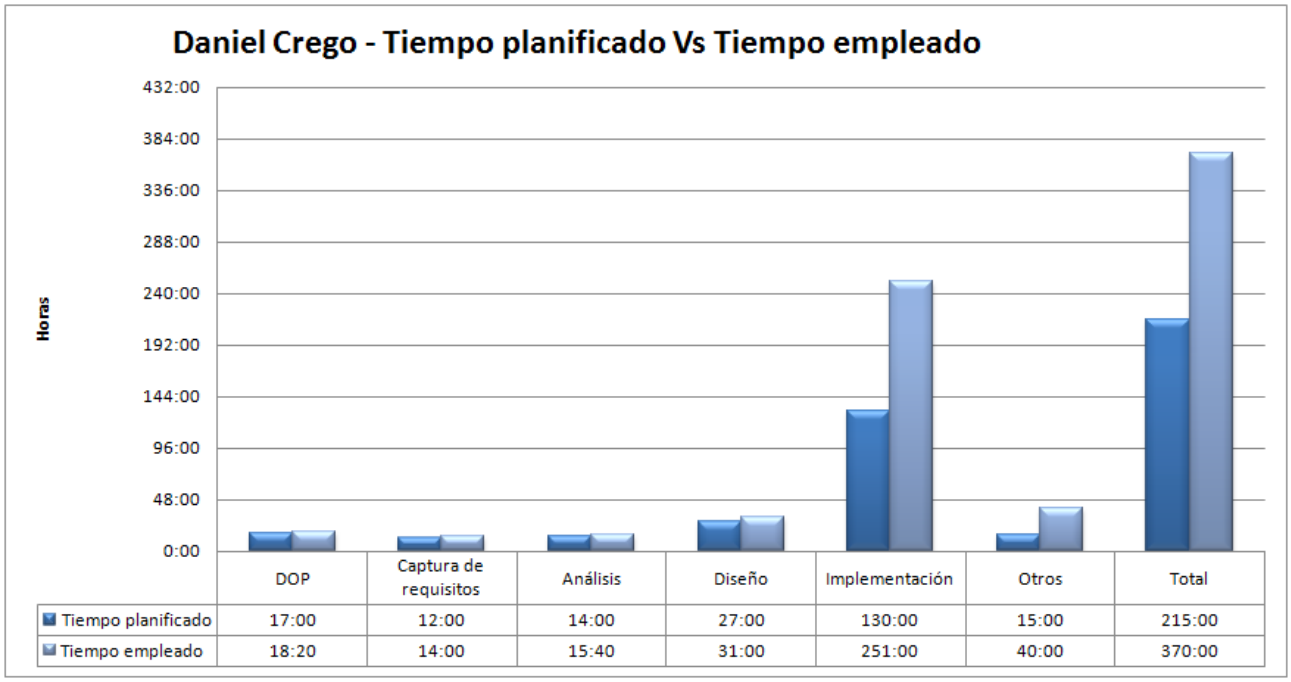
Mientras, la parte lógica se centró en preparar el juego on-line. Esto implicó mucho aprendizaje de los servicios y las bases de datos de Android, también fue necesario adaptar la capa de negocio para incluir el juego on-line. Actualmente nos encontramos en proceso de desarrollo del modo on-line.

Horas reales vs horas estimadas



	DOP	Captura de requisitos	Análisis	Diseño	Implementación	Otros	Total
Tiempo planificado	15:00	12:00	14:00	27:00	130:00	15:00	213:00
Tiempo empleado	15:00	11:40	13:30	26:15	262:15	51:30	380:10

Ilustración 34 - Comparación de tiempo Adrián Gil



	DOP	Captura de requisitos	Análisis	Diseño	Implementación	Otros	Total
Tiempo planificado	17:00	12:00	14:00	27:00	130:00	15:00	215:00
Tiempo empleado	18:20	14:00	15:40	31:00	251:00	40:00	370:00

Ilustración 35 - Comparación de tiempo Daniel Crego

En las ilustraciones 34 y 35 podemos observar como la estimación sobre el tiempo necesario para el proyecto fallo en la parte de implementación y en el tiempo planificado para la investigación.

Tiempo empleado	Tiempo planificado	Diferencia
750 horas	428 horas	322 horas

Esto es el resultado de una planificación ineficiente y ha provocado que un gran aumento en la duración del proyecto.

11.PRUEBAS

En el siguiente apartado trataremos de explicar algunas de las pruebas que hemos realizado durante el desarrollo de este proyecto.

Para este apartado hemos clasificado dos tipos de pruebas, las de caja blanca y las de caja negra que describiremos con más detalle a continuación.

11.1. Pruebas de caja negra

Ilustración 36 – Prueba de caja negra

En las pruebas de caja negra se elige una unidad del programa y dada una lista de entradas y sus correspondientes salidas se presentan dichas entradas y se comprueba que la salida sea correcta.

11.1.1. Mover un barco

En esta prueba comprobamos que un barco puede ser arrastrado colocado, girado y recogido. Un barco no puede estar fuera del tablero o del puerto en ningún momento.
Uno de los fallos que localizamos era que al mover un barco no aceptase la posición y éste quedase suelto. Solucionamos el error obligándolo a volver al puerto si no tiene una posición correcta.

11.1.2. Colocar flota

Al intentar colocar toda la flota de forma aleatoria existe el riesgo de no poder colocar algún barco. Hemos analizado el problema y sucede cuando los espacios dejados en el tablero no permiten colocar un barco grande como el portaaviones. Para evitar éstos problemas primero se intentan colocar los barcos de mayor a menor tamaño. Si aún así no es posible colocar el barco simplemente se ignora y se intenta colocar el siguiente.
Para comprobar su correcto funcionamiento hemos realizado una prueba de caja negra que ha resultado positiva. La secuencia de casos de la prueba incluyen situaciones en las que el portaaviones no tiene sitio, en las que sólo cabe en vertical o en horizontal y por supuesto en las que tiene varios sitios posibles.

11.1.3. Bombardear una casilla

Para comprobar el correcto funcionamiento de esta unidad realizamos una prueba de caja negra. En el resultado no observamos ninguna secuencia con un resultado fuera de los esperado por lo que convenimos que el funcionamiento es correcto.

11.1.4. Recortar la niebla

Este efecto hace que la niebla desaparezca de las casillas que has sido bombardeadas. Se realizaron varias pruebas de caja negra en las que encontramos múltiples fallos. Algunos de los fallos eran debidos a la incorrecta utilización del clipping o a fallos en el momento de utilizarlo.
Se solucionó creando una capa de clipping y aplicándola sólo al pintar la niebla.

11.1.5. Arrastrar la pantalla

Aunque Android ofrece formas de arrastrar la pantalla y cambiar el contenido de esta. Tuvimos que desarrollarlo por nuestra cuenta ya que los gráficos utilizados en algunos momentos de la aplicación no son los de Android.
Para comprobar el correcto funcionamiento de esta unidad se realizaron múltiples pruebas de caja negra hasta que el resultado fue negativo. Algunos de los fallos que se observaron eran la colocación incorrecta del tablero o el movimiento de barcos o casillas.

11.2. Pruebas de caja blanca

En las pruebas de caja blanca se elige una unidad del código (función, clase, módulo...) y se ejecuta hasta que se ha pasado por todas las líneas de código. Pretende encontrar excepciones o flujos no deseados.

11.2.1. Pre-compilador de Java

Esta utilidad de *Java* compila el código mientras escribes informando de errores de compilación antes de ejecutar. Con esto consigues encontrar errores que de otra forma pueden producir efectos indeseados.
Uno de los errores más comunes de los lenguajes fuertemente tipados que localiza el pre-compilador de *Java* es una incongruencia de tipos. En estos casos en los que el tipo requerido y el proporcionado no coincide, el compilador encuentra el error y te informa de que no es posible compilar el código.

11.2.2. Tratamiento de excepciones

Es una característica de *Java* y otros lenguajes que comprueba la corrección del código, pero a diferencia del pre-compilador, esta comprobación se realiza en tiempo de ejecución.
El ejemplo típico suele ser cuando en una unidad se realizan operaciones como la división y se trata con resultados infinitos, si en una prueba se realiza una división entre cero y se intenta manipular el resultado como un número se dispara una excepción. Si analizamos la excepción podemos ver que el tipo se trata de una división entre cero y podemos hacer un trazado de pila y localizar el origen de la excepción.

11.2.3. Debugger

Esta herramienta ofrecida por el entorno de desarrollo (Eclipse, en nuestro caso) permite analizar la ejecución del código línea por línea. Es interesante realizar una depuración cuando el resultado de una prueba del tipo de caja negra, por ejemplo, no da el resultado adecuado y no se tiene una idea clara de cómo se ha producido el error.
El ejemplo más claro de una prueba en la que hemos sido conscientes de la utilidad de la herramienta de depuración es el siguiente.
Tras una actualización del comportamiento de la "IA Normal" hicimos unas pruebas rutinarias de caja negra y obtuvimos un resultado inesperado. La IA no cedía el turno al bombardear una casilla de agua y como resultado una vez que obtenía el turno jugaba hasta que ganaba.
Tras realizar un análisis mediante el debugger localizamos un error de escritura en el código pero no se producía al bombardear agua sino al intentar pasar turno.

11.3. Pruebas de implantación

Este proyecto ha sido testeado en 4 dispositivos móviles Android diferentes y en los 4 ha funcionado a la perfección sin ningún problema:

- Nvsbl P4D Sirius 2.2.1
- Sony Xperia ST23i 4.0.4
- Samsung Galaxy 3 GT-I5800 2.1
- Samsung Galaxy S3 mini GT-I8190 4.1.2

Todos ellos cumplían con los requisitos y se veía la aplicación en la pantalla a la perfección a pesar de que cada uno tiene un tamaño de pantalla y resolución distinta. Estaría bien haberlo testeado en algún tablet y netbook para comprobar su total funcionamiento con cualquier tipo de dispositivo pero no contamos con acceso a ningún dispositivo de este tipo por lo cual no se ha podido realizar. Aunque si que se han probado en la maquina virtual con las especificaciones de tablet y netbook y la aplicación funcionaba perfectamente.

12. CONCLUSIONES

12.1. Objetivos logrados

De los objetivos declarados al inicio de la memoria se han cumplido todos. Algunos de una forma más elegante y otros podrían ser mejorables. Pero esto quedará para una futura versión en la que cambiemos gran parte del proyecto. Por esta razón consideramos que el resultado del proyecto ha sido satisfactorio y que supera lo esperado al inicio de este proyecto pese a los problemas surgidos y a los retrasos sufridos.

Otro objetivo no declarado de este proyecto era aprender cómo funciona y cómo se implementa sobre Android y aunque en un principio apenas utilizamos las funcionalidades que se ofrecen algunos añadidos hechos al proyecto utilizan algunas de estas funcionalidades como los servicios y las notificaciones y aunque todavía queda mucho que aprender ambos integrantes consideramos que hemos aprendido mucho más de lo esperado.

12.2. Valoración personal

Para mi el resultado de este proyecto es altamente satisfactorio. He tenido la oportunidad de programar para uno de los sistemas más utilizados en este momento. He aprendido el funcionamiento del versionado utilizando subversión.

También he aprendido como funciona alguna de las herramientas que utilizo todos los días y he visto la cantidad de posibilidades que se ofrecen. Ademas de eso también he adquirido nuevos conocimientos de *Java* y ya que uno de mis motivos para hacer este proyecto era sobretodo ser capaz de crear una aplicación que funcionara correctamente y aprender un nuevo sistema así que me siento más que satisfecho con el resultado final.

12.3. Mejoras futuras

La versión final de NavyWar nos parece bastante elegante. Sin embargo no descartamos mejorar el proyecto en un futuro. Algunos de estos cambios podrían ser añadir distintos estilos, mejorar lo gráficos incluyendo 3D o cambiar el servidor ya que no sabemos la eficiencia del actual si el número de usuarios aumenta.

En la parte de juego on-line también nos planteamos incluir un sistema de niveles basado en los rangos militares. Estos rangos aumentarían según la experiencia adquirida por el usuario jugando partidas on-line. Las partidas ganadas dan más experiencia que las perdidas aunque aún no hemos decidido el sistema de experiencia.

Además incluiríamos una serie de ambientes desbloqueables que situarían la acción en el desierto, bajo el océano, en el espacio, en el polo norte, en el cementerio de Halloween.

13. BIBLIOGRAFÍA

13.1. Desarrollo de juegos

Amit's Game Programming Information – part of Red Blob Games
(http://www-cs-students.stanford.edu/~amitp/gameprog.html)

Gamedev.net
(http://www.gamedev.net/)

13.2. Desarrollo de juegos para Android

"Desarrollo de juegos para Android"
de Mario Zechner, ISBN-13: 978-84-415-3035-5
Traducción de "Beginning Android Games" por Natalia Rodríguez Blanco

13.3. Materia de programación

Información sobre las APIs de Android
(http://developer.Android.com/develop/index.html)

Foro StackOverFlow
(http://stackoverflow.com)

13.4. General

Wikipedia
(http://es.wikipedia.org)

14. ANEXOS

14.1. Manual de usuario

El objetivo de esta guía es instruir al usuario en el uso del juego en caso de ser necesario. Para comenzar, indicar que se trata de una versión del clásico juego de hundir la flota para dispositivos *Android.* Dispone de varios modos de juego y un menú para seleccionar la partida que desee.

14.1.1. Uso de la aplicación

Ilustración 37 - Menú principal

Al iniciar el juego se muestra brevemente una pantalla de carga y seguidamente el menú principal.

En cualquier momento se puede minimizar y suspender la aplicación utilizando el botón *Inicio* () del dispositivo *Android*. Para reanudar la ejecución bastará con volver a iniciar la aplicación o utilizar el historial de ejecución.

Para terminar definitivamente la aplicación utilizar el botón de *Retorno* (). Al hacer esto aparece una pantalla de confirmación.

14.1.2. Menú de partida

En el menú de crear partida nos aparecen las distintas opciones de juego. Muy fácil, normal y dos jugadores. También podemos regresar a la pantalla de inicio pulsando el botón de retorno.

El nivel de dificultad de la IA "Muy Fácil" está diseñado para dar tiempo al usuario a que tome contacto con las reglas y funcionamiento del juego. Mientras que el modo "Normal" jugará ofreciendo cierta dificultad ya que una vez que encuentre un barco tratará de eliminarlo lo antes posible. En la pestaña oponente humano podemos elegir la opción "2 jugadores", que participan en una partida compartiendo un único dispositivo *Android.*

A partir de este momento empezara el juego. En cualquier momento se puede pulsar el botón de *Retorno* () que abrirá un menú de opciones.

Ilustración 38 - Menú de pausa

En este menú aparecen tres opciones: Volver al menú que cierra la partida actual y vuelve al menú de partida. Reiniciar cierra la partida actual y empieza una nueva del mismo tipo (muy fácil, normal, dos jugadores). Reanudar cierra el menú de pausa y reanuda el juego.

14.1.3. Pantalla colocar flota

Ilustración 39 - Fase de preparación

Una vez seleccionado el modo de juego se solicitará al usuario que coloque su flota, en caso de ser una partida de "2 jugadores" se muestra la pantalla dos veces, una por participante.

La pantalla consta de un tablero de 10x10 casillas donde colocar los barcos, un puerto donde se sitúan los barcos antes de colocarlos, y tres botones: el primero () permite colocar todos los barcos en el tablero de forma aleatoria para ahorrar tiempo. Además cuando todos los barcos están colocados el botón cambia a () para indicar que se ha acabado de colocar los barcos.

El segundo botón limpia el tablero devolviendo todos los barcos al puerto y el tercero cambia la orientación del puerto para poder colocar los barcos en vertical u horizontal a gusto del usuario.

En el puerto se muestra sólo un barco de cada clase pero en realidad hay 10 barcos para colocar:

- 1 Portaaviones (tamaño: 4)
- 2 Destructores (tamaño: 3)
- 3 Fragatas (tamaño: 2)
- 4 Submarinos (tamaño: 1)

Para colocar un único barco basta con arrastrarlo hasta la posición deseada y para recogerlo tenemos que arrastrarlo hasta el puerto.

Nótese que al colocar un barco tanto las casillas que ocupa como las que le rodean cambian ligeramente de color, esto indica que esas casillas no pueden ser ocupadas por otro barco.

Una vez colocados todos los barcos el icono del primer botón cambia para permitirte continuar con la partida.

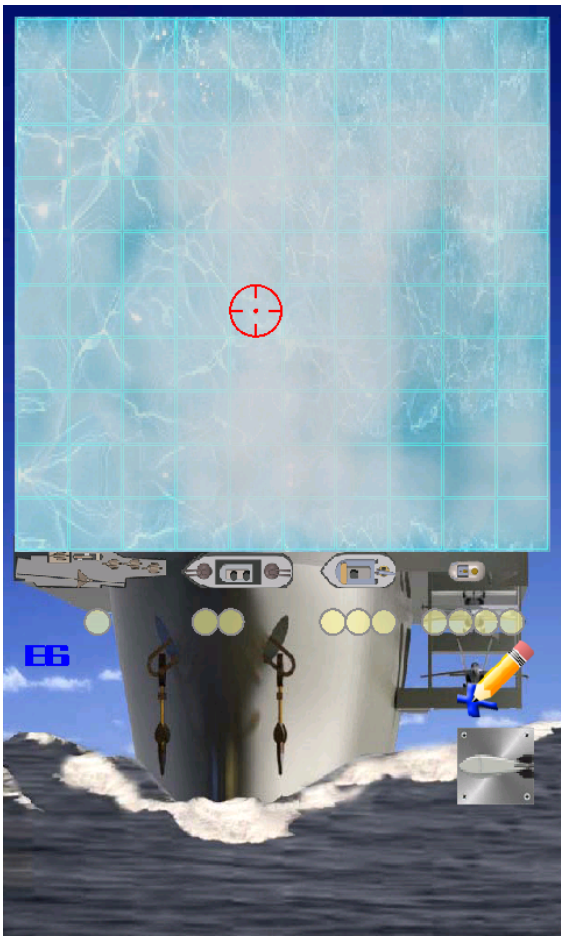


Ilustración 40 - Fase de combate

14.1.4. Pantalla jugar partida

Cuando los dos jugadores han terminado la fase de preparación (la IA coloca su flota automáticamente) se pasa a la en la que se muestra el tablero del oponente cubierto de niebla de guerra (significa que se desconoce lo que hay debajo) que se irá disipando casilla a casilla a medida que se bombardee el tablero del oponente.

Justo por debajo hay un indicador que marca el estado de la flota. Si el barco esta hundido el circulo es de color rojo y si no de color amarillo.

Al arrastrar a la derecha aparece el tablero propio para poder comprobar el progreso del oponente ademas del estado de tu flota. También hay dos botones, el primero sirve para cambiar entre el modo "Bombardear", que permite apuntar a una casilla y lanzar un misil, y el modo "Marcar", que permite marcar o desmarcar una casilla. Al seleccionar una casilla con el modo "Marcar" activo aparece una X que impide bombardear esa casilla, si ya estaba marcada la X desaparece. El segundo botón permite lanzar una bomba o pasar turno si el último resultado es agua.

En una partida de 2 jugadores al principio del turno se muestra una pantalla que tiene una doble función: Informar de quién tiene el turno y evitar que los participantes vean el tablero de su oponente.

El proceso a seguir para bombardear una casilla es el siguiente:

1. Situar el objetivo sobre una casilla cubierta por la niebla con un toque.
2. Lanzar la bomba con un segundo toque en la casilla con el objetivo o con el botón de la bomba.

Cuando la bomba impacta en la casilla despeja la niebla revelando lo que había debajo:

- Agua: La bomba no ha tocado ningún barco enemigo y te toca terminar tu turno.
- Tocado: La bomba impacta en un barco pero éste aún es capaz de seguir a flote. El caos causado en la flota enemiga te permite continuar disparando.
- Hundido: La bomba destruye el barco enemigo ocasionando una onda expansiva que despeja la niebla alrededor de éste. El caos causado en la flota enemiga te permite continuar disparando.

Cuando el resultado es agua hay que indicar el fin de turno tocando el botón de la flecha (), para que el oponente pueda jugar.

En cualquier momento puedes acceder al menú de pausa utilizando el botón de retorno.

Cuando un jugador haya destruido por completo la flota de su oponente aparece un aviso indicando si se ha ganado o perdido. También aparece el menú de pausa con la opción reanudar cambiada a salir. Si pulsamos esta opción nos aparece un aviso de confirmación y se cierra la aplicación.

14.2. Prototipo III: On-line

El último prototipo incluye una capacidad de juego on-line(en red), que permitiría al usuario disfrutar de unas partidas con otros usuarios a través de internet sin la necesidad de compartir un mismo dispositivo.

La primera decisión a la que nos enfrentamos fue la estructura de comunicación. Nos debatíamos entre una clásica estructura Cliente-Servidor o una P2P. Ambas tienen sus ventajas e inconvenientes.

Cliente-Servidor	P2P
Servidor: • Tráfico elevado (todas las operaciones).	Servidor: • Tráfico reducido (interconexión).
Base de Datos: • Mayor control. • Menor riesgo de inconsistencia.	Base de Datos: • Menor control. • Mayor riesgo de inconsistencia.

• Necesidad de backup externa.	• Backup implícita.
Cliente: • Transferencia de datos ligera.	Cliente: • Transferencia de datos pesada (todas las operaciones).
Conexión: • Depende del estado del servidor.	Conexión: • Requiere ambos Peer conectados.

Finalmente nos decidimos a utilizar la estructura cliente servidor, lo cual implica una nueva aplicación para el servidor que incluye una base de datos, una interfaz de conexión y otra de control. Además adaptar el cliente para permitir y comprender la comunicación con el servidor.

Debido a un presupuesto reducido decidimos utilizar un hosting gratuito como servidor lo cual nos limita a un servidor vía HTTP. La mayoría soportan PHP y base de datos SQL, de modo que el desarrollo del servidor se lleva a cabo utilizando dichos recursos.

La base de datos SQL no es excesivamente compleja estructuralmente hablando pero para aprovechar el soporte para transacciones paralelas hemos tenido que implementar las operaciones en SQL.

Por encima se encuentra la interfaz del servidor que conecta con las operaciones de la base de datos y para transmitir información hemos convenido utilizar un estándar XML con etiquetas propias. La implementación de dicha interfaz se lleva a cabo en PHP.

El cliente por su parte también proporciona una interfaz que consume el documento XML y lo traduce a objetos Java.

Finalmente para tener una conexión que se mantenga actualizada y ya que HTTP se basa en TCP necesitamos una estrategia de encuesta la cual se la encargamos a un servicio que forma parte de nuestra aplicación y que se mantendrá activo aunque se finalice la actividad principal.

14.3. Guía de Instalación

Antes de nada, indicar que esta aplicación solo puede instalarse en dispositivos Android 1.5 o superior.

Puedes instalar la aplicación desde varias fuentes diferentes, a continuación detallaremos las que no requieren el código fuente.

Después de instalar el juego tal vez desee establecer un acceso directo desde uno de sus escritorios.

14.3.1. Instalación desde Google Play

Accede a Google Play con tu cuenta desde un navegador en la siguiente dirección: http://play.google.com/store o utiliza la aplicación para dispositivos Android.

Busca la aplicación por el nombre "NavyWar" o utilice el enlace: http://play.google.com/store/apps/details?id=com.navywar

Pulse el botón instalar, a continuación se le indicarán los permisos que requiere la aplicación, compruébelos con detenimiento siempre.

Tras aceptar los permisos espere a que la aplicación se descargue e instale.

14.4. Guía para desarrolladores

Para instalar esta versión lo único que tendremos que hacer es acceder a la dirección http://developer.android.com/intl/es/sdk/index.html y descargarnos el paquete de instalación, este consiste en un .zip con dos carpetas, una para eclipse y otra para el SDK y un .exe. Descomprimimos estos archivos en la ubicación donde deseamos tener nuestro Eclipse, para mayor comodidad será recomendable crear un acceso directo del ejecutable que se encuentra en la carpeta eclipse. Si ejecutamos ese archivo se iniciara eclipse con el plugin de Android ya instalado y ya tendremos todo preparado para empezar a desarrollar en Android.

14.5. Google Analytics

Añadiendo este complemento a nuestra aplicación obtendremos multitud de información sobre el uso de nuestra aplicación, desde zonas donde se utiliza, versiones de Android que lo usan, cantidad de usuarios y usuarios en tiempo real.