FIUBA - 7507

Algoritmos y programación 3

Trabajo práctico 1: Algo42

1er cuatrimestre, 2011

(trabajo individual)

Fecha de entrega: viernes 22 de abril

Nombre: Bárbara Aguilá Cudicio

Padrón: 92071

Email: aguila.cudicio@gmail.com

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando individualmente utilizando Smalltalk.

Consigna

Desarrollar el modelo de clases, con sus pruebas correspondientes, de un juego cuyas reglas se muestran a continuación, y se pedirá pruebas unitarias y de integración completas, y elementos de documentación que se detallan en este enunciado.

Introducción

La empresa Rezagos Militares Software Inc., creadora de grandes éxitos en el campo de juegos de video, está encarando un proyecto para realizar un simulador de batalla de aviones para ser utilizado como software de entrenamiento de la Fuerza Aérea Argentina y ha elegido a su grupo de trabajo para realizar el diseño y desarrollo de su nueva e innovadora idea.

El juego consiste en un avión de combate que debe combatir naves enemigas.

Los escenarios deben tener 2 componentes principales: el fondo y los elementos móviles, como aviones enemigos, armas enemigas, objetos especiales, el avión del jugador y las armas del jugador.

A continuación se describe el enunciado general con las características funcionales de la aplicación a desarrollar:

> Corre el año 2042 y nuestro país debe defenderse de una invasión extranjera que busca el control de las fuentes de agua potable de nuestras provincias. Nuestra flota aérea consta de 2 aviones, uno de los cuales no funciona por falta de mantenimiento. La flota extranjera es muy poderosa y está compuesta de miles de aviones que comienzan a sobrevolar nuestro territorio y amenazan con controlarlo por completo. Pero aun queda una esperanza si nuestro único avión (cuyo nombre clave es “Algo42”) pudiera llegar hasta el porta-aviones enemigo y arrojarse en picada sobre él, destruyendo el cuartel de control de los invasores.
>
> Para cumplir con su cometido, el Algo42 deberá cumplir una serie de misiones. En cada misión se enfrentara a una flota distinta de aviones invasores. Las flotas de los enemigos están conformadas por distintos tipos de aviones.
>
> La cantidad de aviones de las flotas es variable (mínimo 15 aviones). Cada flota cuenta con un avión Guía que coordina al resto de los

aviones de la flota. En caso de destruirse el avión Guía los demás aviones detienen sus disparos instantáneamente y huyen del campo de batalla.

Una implementación de una empresa competidora puede verse en el siguiente link, y sirve para entender la dinámica del juego:
http://www.youtube.com/watch?v=xQIB-O0DZm4
Los enemigos cuentan con los siguientes modelos de naves:

Nombre	Armas	Estrategia de vuelo	Observaciones	Puntos por destrucci on
Avionetas	Lasers	Idas y vueltas en línea recta	Son los aviones más rápidos.	20
Bombarder os	Lasers, cohetes y torpedos rastreadores	Zig/Zag	Son los más poderosos pero al mismo tiempo los más lentos. Al ser destruidos, el Algo42 puede tomar sus armas.	30
Explorador es	No tiene	En círculos.	Vuelan en círculos amplios recorriendo toda la superficie aérea, en búsqueda de chocar al Algo42.	50
Cazas	Torpedos simples	En grupo formando una V	Al ser destruido su tanque de energía pude ser tomado por Algo42.	30

Por su parte el Algo42 es un avión escalable. En la versión base solo cuenta con lasers, pero puede escalar aumentado su poderío apropiándose de las armas y energía de los aviones que destruye.

Consideraciones generales:

- Todo avión tiene una fuente de energía, la cual disminuye a medida que es atacado. Cuando dicha energía llega a cero el avión es destruido.
- El Algo42 va sumando puntos para su misión a medida que destruye aviones enemigos. Al llegar a 1000 puntos termina el nivel y pasa al siguiente.
- Los lasers no se gastan, pero los torpedos y cohetes sí.
- El espacio no esta vacío, además de las flotas enemigas hay aviones civiles (pasan en línea recta a poca velocidad, el Algo42 debe evitar destruirlos ó chocarlos, caso contrario pierde 300 puntos por cada avión civil destruido) y helicópteros de la policía federal (se mueven en círculos pero tienen orden de no disparar, también debe evitarse su destrucción o se pierden 200 puntos por cada helicóptero).

Supuestos

Las naves enemigas salen del juego cuando están fuera de los límites del plano. Esa decisión fue tomada a partir del video propuesto como muestra de la dinámica del juego.

Algo42 no es una nave única. Una de las propuestas que surgieron en clases fue hacer que algo42 sólo pudiera instanciarse una vez. Esto no fue realizado, ya que una implementación así no permite futuras implementaciones para dos jugadores.

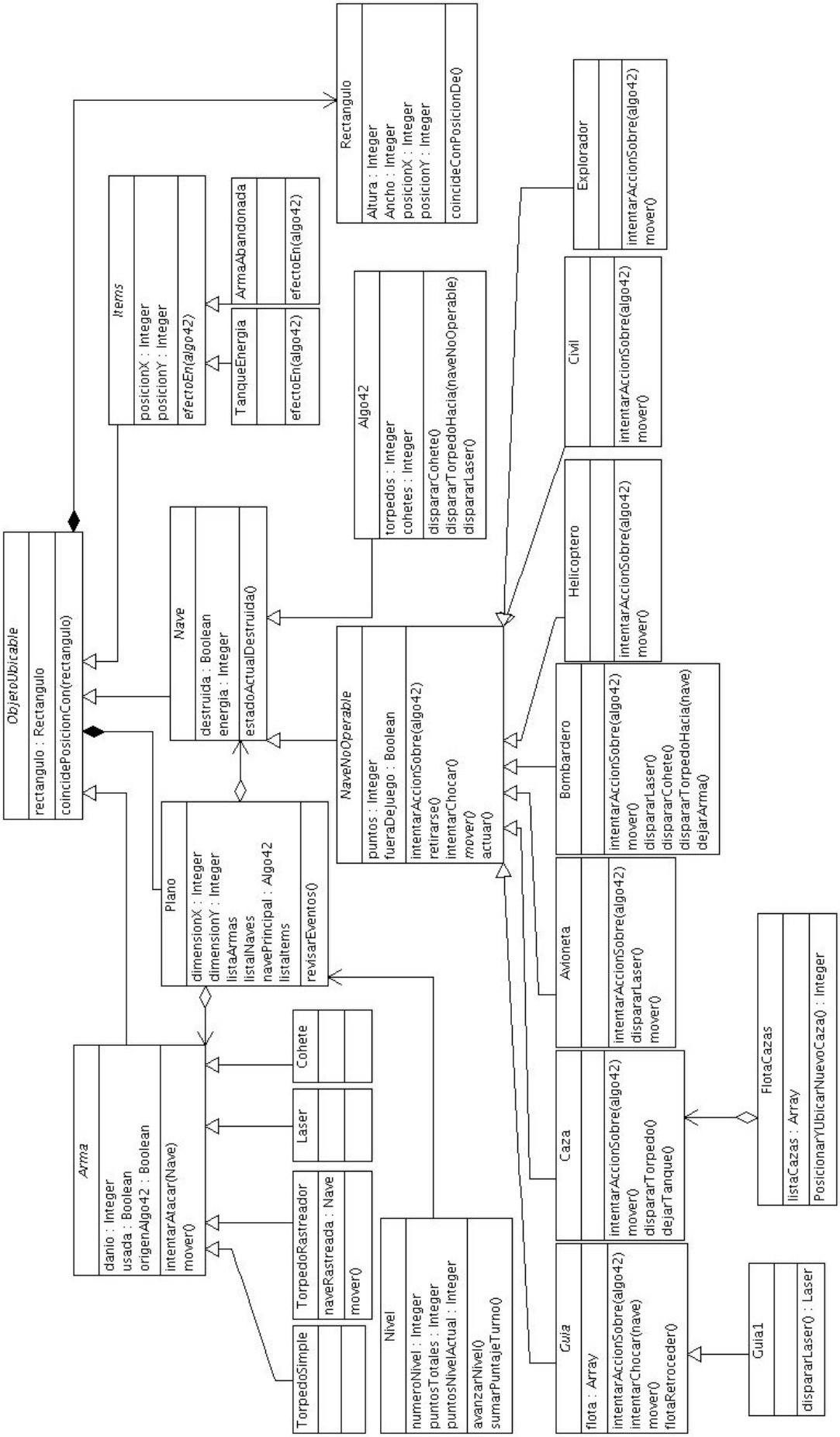
Las naves guias no pueden ser chocadas. De esta manera, la complejidad del juego incrementa un poco. Parece lo más lógico que las naves guias tengan características que las hagan más díficiles de eliminar.

Las demás naves enemigas se destruyen al ser chocadas.

Las naves no operables no pueden chocarse entre ellas. Cambian la dirección de su movimiento para no chocarse.

Modelo de dominio

Diagramas de clases



El diagrama de clases no incluye absolutamente todos los métodos y atributos de las clases: Solamente los más importantes. Fue excluida del gráfico toda la información que no fuese fundamental para entender el funcionamiento de las clases; Especialmente, fueron ignorados los métodos para modificar atributos o devolverlos (“getters” y “setters”), al igual que varios atributos que servían a las instancias de naves no operables para determinar la posición en la que tenían que ubicarse al llamar al método “mover()”.

Detalles de implementación

Rectangulo y ObjetoUbicable: Aclaraciones sobre como se representan los objetos en el espacio.

Una instancia de rectángulo modeliza absolutamente todos los elementos que ocupan un lugar en el espacio. Es inicializado con un ancho y un alto determinados, y una posición en el plano. Para entender bien las comparaciones que se realizan entre las posiciones de los ObjetosUbicables, es importante destacar que las posiciones x e y que el rectángulo reconoce como propias corresponden a la ubicación del punto extremo izquierdo inferior. Así, para llevar a cabo una comparación entre objetos, deben compararse sus rectángulos. Si comparten al menos un punto en común, el método coincidePosicionCon: otroRectangulo devolverá true. Para tal fin se comparán los extremos de ambos, y se realizarán algunos cálculos sencillos para determinar si ambos se superponen.

También, cuando se mueven los objetos, lo que cambia es la posición de su punto extremo izquierdo inferior. Todos los otros puntos que ocupan son determinados cuando son necesarios, a partir del área y el ancho de sus respectivos rectángulos, que son constantes.
Cada objeto ubicable, entonces, contiene referencias a dos objetos fundamentales: Un plano y un rectángulo. El rectangulo es único y propio de un sólo objeto -Es decir, cada instancia de NaveNoOperable tendrá una instancia de rectángulo que no compartirá con otros objetos- pero el plano será común para todos los elementos del juego. Que cada objeto ubicable tenga referencia al plano es muy importante, ya que eso le permite realizar sus movimientos: La nave necesita saber si la posición que ocupa es compartida con otro objeto, para así saber si se está dando una colisión, y también para saber si se está saliendo del área de juego.

Plano: En el item anterior se introdujo a la clase Plano. Básicamente, su estado tiene referencias a todas las listas de objetos del juego -items, naves y municiones-. Su método revisarEventos se encarga de iterarlas, para que cada objeto pueda llevar a cabo su comportamiento. También, determina en su estado los limites del plano del juego. Diferentes objetos reaccionan de distinta manera ante estos límites. Las naves enemigas y las armas quedan “Fuera de juego” al dejar estos límites, al igual que las armas. Estos objetos serán borrados de las listas antes de terminar la ejecución del método revisarEventos. En cambio, una instancia de algo42 jamás podrá quedar fuera de juego, ya que eso levantaría una excepción.

Aclaraciones sobre la representación de las colisiones Con las clases Plano, Rectangulo y ObjetoUbicable, se pudo llevar a cabo la resolución y análisis de las colisiones. La idea es la siguiente: Todos los objetos se representan como rectangulos. Sabiendo sus medidas y sus posiciones en el espacio, sencillos algoritmos con unos pocos cálculos pueden determinar si dos rectángulos se están superponiendo o no. Esto puede ser tomado como una colisión. Y con colisiones, no sólo nos referimos a colisiones entre naves: También a colisiones entre naves y municiones o el encuentro entre la nave del jugador y un item dejado en el campo por una nave enemiga.
Para detectar una colisión, cada nave debe revisar todas las listas de elementos contenidos en el plano cada vez que cambia de posición. Su comportamiento dependerá de qué tipo de elemento encuentre: Por ejemplo, si es una nave enemiga bajará sus puntos de energía; si es un item del tipo Tanque, en cambio, los incrementará, etc. Los encargados de detectar las colisiones son las demás naves, items y armas, no el algo42. El jugador sólo se encarga de decirle al algo42 que se mueva o que dispare. Los demás elementos, al recibir el llamado de la función actuar, deciden cual debe ser su comportamiento. Si un movimiento suyo o del algo42 provoca que se superpongan, serán ellos los encargados de detectarlo, con la función intentarChocar.

Una vez que se detectó una superposición entre dos elementos, cada objeto decide qué hacer. Por ejemplo, como ya se mencionó antes, dos naves no operables cambian su sentido para evitar chocarse. Si se trata de una nave y una munición, la munición analizará si el elemento con el que se superpone es del mismo bando que la originó: Es decir, las armas guardan una variable de tipo booleano en su estado, que les ayuda a determinar si fueron lanzadas inicialmente por un algo42 o una nave no operable. De esta manera, se evitan situaciones tales como dos naves no operables reduciendo puntos de sus tanques de energía en ataques dentro del mismo bando, o aún peor, que un algo42 se ataque a si mismo. En el caso de que se encuentren en una superposición un algo42 y un ítem, este último tiene un método aplicarEfecto que realizará algún tipo de comportamiento sobre la nave del jugador, dependiendo de qué tipo de ítem sea.

NaveNoOperable: Este grupo esta conformado por todas las naves que el usuario no puede manejar. Es una clase abstracta, por lo cual las verdaderas instancias heredan de ella. Su método “actuar” le dice a cada instancia en particular cuál es su función. Todas las naves intentan moverse o chocar al algo42 cuando esta función es llamada; Algunas naves, como el bombardero, también dispararan.

Puesto que las diferentes clases hijas de NaveNoOperable tienen distintos tipos de movimiento, también cada una de ellas cuenta con un método movimiento alternativo distinto. Esto es porque el plano está diseñado para no permitir que dos naves no operables ocupen el mismo lugar ni se superpongan en ningún punto; Es decir, las naves no operables no pueden chocarse entre ellas. La función intentarMovimiento es la encargada de manejar las llamadas a los métodos de movimiento. Primero se llama a mover, que maneja el movimiento por defecto, y luego a moverAlternativo. Podría suceder, como se muestra en algunos ejemplos entre las pruebas del presente trabajo, que la nave se

encuentre rodeada y no pueda escapar con ninguna de sus dos formas de movimiento. En ese caso, no cambia su posición. Otro punto a destacar, respecto a este tema, es que los movimientos alternativos siguen la descripción del movimiento por defecto. Cambian su sentido, pero la idea sigue siendo la misma. Por ejemplo, el vuelo de la avioneta sigue siendo rectilíneo, el del bombardero en zig zag, el del explorador en círculos, etc.

Guia: Las naves Guias no son instancias de clases iguales, sino que cada jefe de flota debería heredar de la clase abstracta Guia. De esta manera, para desarrollar varios niveles, podrían crearse diferentes jefes que presenten mayores dificultades. La clase guía (Guia1) implementada es muy simple, y solamente consiste en un ejemplo para mostrar como sería su instanciación y utilización.

Excepciones

AlgoSeAtacaASiMismoError: Esta excepción debe ser lanzada cuando una munición lanzada por una nave instancia de Algo42 intente atacar a la misma nave que le dió origen.

AreaInvalidaError: Debe ser lanzada cuando una instancia de Algo42 realice un movimiento invalido, por el cual quedaría fuera del área de juego. En las pruebas se muestra una solución para capturar este error, por la cual cada vez que la nave llame métodos que la desplacen a una zona invalida, se optará por prohibirle ese movimiento y dejarla en la posición anterior en la que estaba. También se levantará este error si se intenta crear una instancia de algo42 en un área invalida.

ArmaNoDisponibleError: Será lanzado cuando una instancia de Algo42 intente utilizar un arma que no tiene.

ArmaNoUsadaError: Este error se levantará cuando, teniendo un arma cuyo estado indique que no fue usada, se intente añadirla a la lista de armas usadas de la clase Plano.

ArmaUsadaError: Muy similar al error anterior, pero en el caso contrario; Se levantará cuando, teniendo un arma cuyo estado indique que fue usada, se intente añadirla a la lista de armas en juego de la clase Plano.

AtaqueEntreNavesNoOperables: Servirá para indicar error cuando una munición lanzada por una nave no operable intente reducir el tanque de energía de otra nave de este tipo. Esto nunca ocurre si se usa la clase Plano y se respeta su sistema. De todas maneras, puede suceder si se llaman instancias de naves y sus métodos directamente.

GuiaNoDestruidaError: Será lanzada si las naves pertenecientes a una flota enemiga intentan retirarse cuando su nave guía aún no está destruida.

ItemNoDisponibleError: Será lanzado si una instancia de nave no destruida intenta dejar un arma.

ItemNoUsadoError: Este error se levantará cuando, teniendo un item cuyo estado indique que no fue usado, se intente añadirlo a la lista de items usados de la clase Plano. ItemUsadoError se encargará del caso inverso.

NaveARastrearError: Este error se levantará si el jugador intenta elegir a su

propia nave algo42 como objetivo de su torpedo rastreador.

NaveDestruidaError: Este error ocurrirá si se intentara agregar una nave destruida a la lista de naves funcionales de la clase Plano. NaveNoDestruidaError, nuevamente, realiza la misma tarea pero a la inversa.

NivelError: Será lanzado si se intenta avanzar de nivel teniendo menos de 1000 puntos.

SuperposicionNavesError: Será lanzado si se crearan dos naves no operables en la misma posición, o si se movieran de tal forma que provocaran la superposición de sus figuras.

Diagramas de secuencia

Diagrama de Choque

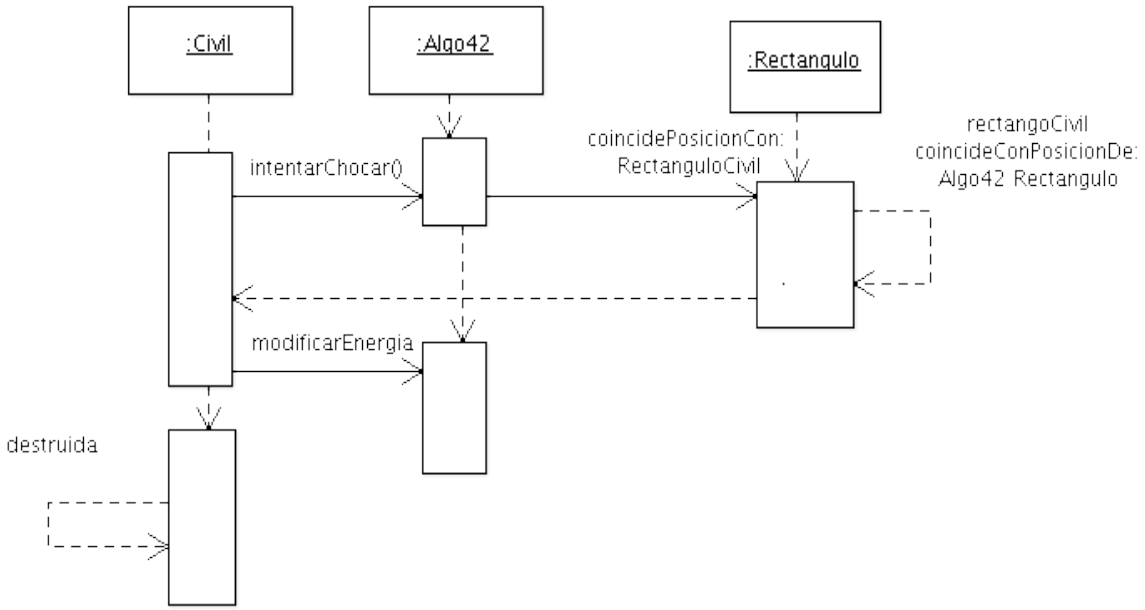
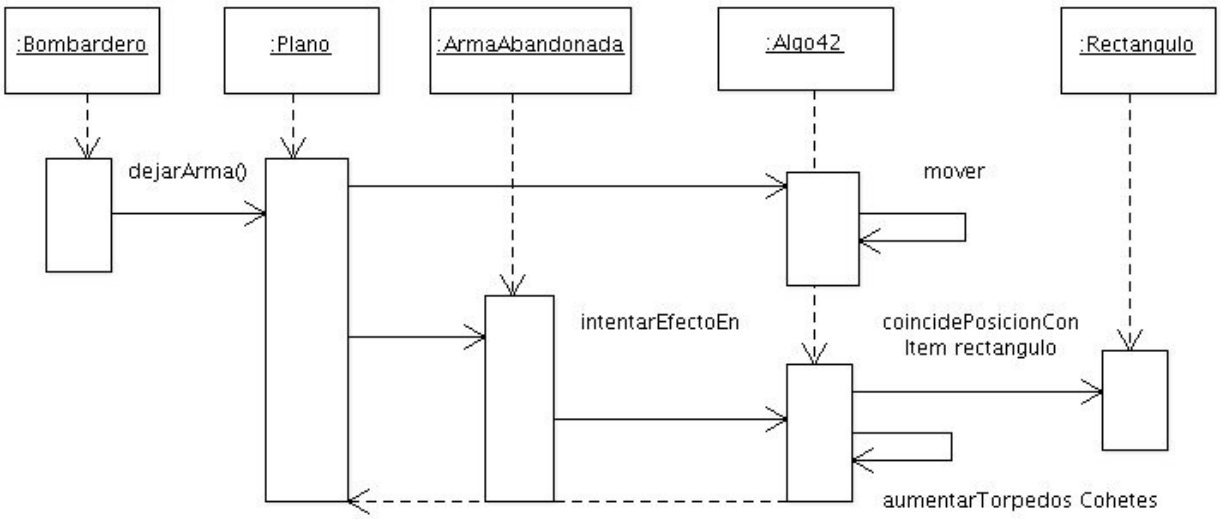


Diagrama de Items



Checklist de corrección

Esta sección es para uso exclusivo de la cátedra, por favor no modificar.

Carpeta

Generalidades

□¿Son correctos los supuestos y extensiones?

□¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

□¿Está completo? ¿Contempla la totalidad del problema?

□¿Respeta encapsulamiento?

□¿Hace un buen uso de excepciones?

□¿Utiliza polimorfismo en las situaciones esperadas?

Diagramas

Diagrama de clases

□¿Está completo?

□¿Está bien utilizada la notación?

Diagramas de secuencia

□¿Está completo?

□¿Es consistente con el diagrama de clases?

□¿Está bien utilizada la notación?

Código

Generalidades

□¿Respeta estándares de codificación?

□¿Está correctamente documentado?