UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
I.T. Informática de Gestión
Proyecto Fin de Carrera

Desarrollo de un escenario en el entorno Greenfoot: Hundir la flota

Autora: Dña. Sara Tena García

Directora: Prof. Susana Montero Moreno

Septiembre, 2009

Índice

Índice de figuras

Índice de tablas

Glosario de términos

Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

IDE: *Integrated Development Environment* es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

RBS: diagrama jerárquico que representan los recursos humanos y materiales de un proyecto.

SLOC: *Sources lines of code,* es una métrica utilizada para medir el tamaño de un programa de software, contando el número de líneas de texto del código fuente del programa. Se suele utilizar para predecir la cantidad de esfuerzo que se requiere para desarrollar un programa, así como para estimar la productividad de programación o el esfuerzo una vez que se produce el software.

UML: *Unified Modeling Language,* es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

1. Introducción

Este documento es parte co-principal del proyecto 'Desarrollo de un escenario en el entorno Greenfoot: Hundir la flota' concebido como Proyecto Fin de Carrera por el departamento de Informática de la Universidad Carlos III de Madrid.

El propósito es transmitir al lector de una forma clara y formal, a través de las distintas secciones que componen el documento, la causa y efecto del desarrollo de dicho proyecto como innovación docente para facilitar el aprendizaje de la orientación a objetos en java mediante la utilización del entorno de desarrollo Greenfoot.

1.1. Planteamiento del problema

La orientación a objetos es uno de los paradigmas más importantes y significativos de los últimos años. Sus técnicas pueden aplicarse durante el análisis, diseño e implementación de los programas, y permiten enfocar el problema que se quiere resolver en términos similares a los utilizados por la mente humana.

Es la enseñanza de la programación orientada a objetos como primer lenguaje la que desencadena el principal problema de éste proyecto. Los estudiantes antes de poder realizar un programa sencillo deben tener una clara noción de lo que es una clase, un objeto, un método, un paquete, etc. Y son éstos precisamente los que generan dicha problemática.

Estudios realizados por el Instituto Científico Weizmann a cerca de la enseñanza temprana de la orientación a objetos han demostrado que las principales causas que dificultan la comprensión de la programación orientada a objetos son los conceptos de clase y objeto, debido a que los estudiantes no lo diferencian. Así como el estado de un objeto, el paso de mensajes, el hecho de que un método puede cambiar el estado de un objeto, etc. [Ben-Ari, Ragonis, Ben-Bassat, 2002].

Es por ello que existen numerosos manuales y herramientas creadas con el único fin de facilitar la comprensión de los conceptos básicos de la programación orientada a objetos, pues no es suficiente aprender la sintaxis de un lenguaje como es Java para poder resolver un problema, sino que se deben tener claros los conceptos para poder llevarlo a cabo.

Sin embargo a pesar de la gran cantidad de manuales y herramientas que existen, no hay certeza de un método que sea 100% eficaz. Es de gran utilidad la

visualización de los programas para facilitar la comprensión de los conceptos citados anteriormente.

1.2.Objetivos

Una vez planteada la existencia de una problemática en el ámbito de la programación orientada a objetos, se puede definir de forma global que el objetivo principal del presente proyecto es **proporcionar un mecanismo visual que permita la representación de ciertos conceptos de la orientación objetos en Java mediante la simulación de un escenario**.

Este objetivo engloba otras cuestiones más específicas como son:

- **Diferencia entre clase y objeto:** Distinguir entre clases, objetos y constructores puede ser confuso para alumnos que se inician en la programación orientada a objetos. Pero entender las diferencias entre estos términos, fundamentales de la programación Java es importante para saber cuando estamos definiendo una clase, cuando estamos creando un objeto, y cómo se construyen dichos objetos.
- **Invocación de métodos:** Con este objetivo se pretende que los alumnos comprendan, cómo para indicar a un objeto que realice una tarea es necesario enviarle un mensaje y cómo para que un objeto procese el mensaje que recibe, la clase debe poseer un método que coincida con ese mensaje. Se pretende de esta manera que los alumnos interactúen con los objetos modificando y accediendo a su estado, etc.
- **Propiedades de los objetos:** Distinguir las propiedades de un objeto: el estado, el comportamiento y la identidad.

Por otro lado, destacar los objetivos personales de la realización de este Proyecto Fin de Carrera:

- **Conceptos de la programación orientada a objetos**: fijar los conceptos de la programación orientada a objetos de los cuales se recibe una leve introducción a lo largo de la carrera. Conceptos muy importantes para el mundo laboral y poco valorados en la enseñanza de Ingeniería Técnica en Informática de Gestión.
- **Java:** el aprendizaje y utilización de la sintaxis y principales características del lenguaje Java.

1.3.Metodología

La satisfactoria consecución de todo proyecto de desarrollo software implica la utilización de una metodología de trabajo concreta, sistemática y predecible, la cual debe adaptarse además a las características del proyecto. Con estos mismos objetivos, sea definido inicialmente la metodología más adecuada a seguir para la obtención del producto final del Proyecto Fin de Carrera. Esta se presentará a continuación:

- **Estudio del problema y análisis de requisitos:** se presenta el contexto de realización del trabajo realizando una revisión de las tecnologías, herramientas y trabajos previos realizados sobre el mismo. Se plantean tras esto los objetivos, y requisitos de dicho proyecto, así como una descripción en profundidad de la problemática que ha dado origen al desarrollo del proyecto.
- **Diseño e implementación:** se plantean las distintas fases asociadas a la construcción de la solución: análisis, diseño e implementación. Se hará uso de las técnicas más adecuadas para la correcta evolución de cada una de las fases, incluyendo los productos (modelos, gráficos, tablas, etc.) que deriven de dichas fases.
- **Evaluación de la solución:** se demostrará la validez de la solución elaborada. La solución se considerará válida si resuelve los problemas expuestos en el planteamiento del problema y satisface los objetivos definidos en la introducción.
- **Documentación:** contendrá todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML), diagramas, pruebas, manuales de usuario, etc.; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.

1.4.Estructura del trabajo

En este apartado se pasa a describir a grandes rasgos la estructura del presente documento. Consta de siete capítulos principales en los que se describe la información asociada al desarrollo del proyecto. Esto capítulos se completan con una serie de anexos con información adicional que de notable relevancia.

Los contenidos que presenta cada capítulo se enumeran a continuación:

1. **Introducción**: se realizará una breve presentación del proyecto explicando tanto el planteamiento del problema, como sus objetivos y la metodología de trabajo utilizada para llevarlo a cabo.
2. **Estado de la cuestión**: se comentará la situación actual de las tecnologías relacionadas con el proyecto y las herramientas analizadas previamente para la realización del proyecto.
3. **Gestión de proyecto software**: consistirá en una estimación del tiempo del proyecto, el presupuesto del mismo, la planificación real del proyecto y la gestión de recursos.
4. **Planteamiento del problema y solución**: se proporcionará un resumen de la problemática que ha llevado a la realización del proyecto, los requisitos planteados por el usuario, la solución que se ha planteado para solucionar la problemática evaluando la adaptación de las herramientas estudiadas con los requisitos del usuario, el trabajo que se deberá desarrollar y la arquitectura de la aplicación.
5. **Análisis**: se profundizará en la explicación del desarrollo del proyecto. Formarán parte de este capítulo el análisis de requisitos que ha de cumplir la aplicación.
6. **Diseño**: mostrará el diseño de la aplicación mediante diagramas de clases.
7. **Implementación**: se centrará en los problemas y cuestiones del desarrollo de la aplicación, mostrando todos aquellos diagramas que se consideren necesarios para la comprensión del citado desarrollo.
8. **Pruebas**: recoge todas aquellas pruebas llevadas a cabo para comprobar que la solución desarrollada cumple la funcionalidad planteada por el usuario.
9. **Conclusiones:** contendrá un resumen del trabajo realizado, las aportaciones y se analizarán las conclusiones obtenidas tras el desarrollo y documentación del proyecto.

2. Estado de la cuestión

En este capítulo se planteará de forma más específica el contexto en el que se encuadra el proyecto, realizando una revisión de las tecnologías, herramientas y trabajos realizados para ofrecer una solución a la idea genérica que presenta el problema.

2.1. Enseñanza de Java como primer lenguaje de programación

Debido a su popularidad en el contexto de aplicaciones Web y la facilidad con la que los principiantes pueden producir programas gráficos, Java se ha convertido en uno de los lenguajes más usados en los cursos de introducción a la programación.

Algunos de los principales problemas que se han encontrado en la enseñanza de Java son:

- Aproximación a la orientación a objetos
- Problemas con la visualización de la orientación a objetos
- Herramientas de ayuda a la enseñanza de la orientación a objetos

Java puede ser utilizado como primer lenguaje de programación siempre y cuando antes de estudiar dicho lenguaje, se forme al alumno en el desarrollo de la lógica. Pues el principal problema empieza en el momento en que un futuro programador no es capaz de alcanzar una clara comprensión del problema a resolver (análisis), ni identificar los conceptos claves implicados (diseño) para finalmente expresar la solución en un programa (implementación). [Dewar, 2008]

Lo fundamental al aprender a programar es desarrollar la lógica necesaria para solucionar problemas en forma algorítmica, independientemente de algún lenguaje de programación. Para ello se considera necesaria una metodología adecuada.

Esta metodología debe ser el resultado de la integración y adaptación de varias técnicas, como son: los conceptos y estructuras de la programación orientada a objetos, con los diagramas de clases de UML (Unified Modeling Language) y algunos conceptos introducidos por el lenguaje Java.

Dicha metodología permitiría diseñar programas o algoritmos orientados a objetos y preparar a los estudiantes para que puedan aprender y comprender cualquier lenguaje orientado a objetos como Java.

En caso de no promover la resolución de problemas y pensamiento en profundidad se produce un estilo de programación de ensayo y error e impide que los estudiantes adquieran la disciplina de la programación, convirtiéndose únicamente en codificadores.

Aunque existen críticas al uso de Java como primer o único lenguaje de programación, éste tiene un importante rol en la instrucción de los programadores.

2.1.3. Aproximación a la orientación a objetos

La investigación del estudio de una primera aproximación a la orientación a objetos está basada en la premisa por la cual los estudiantes deberían estudiar programación orientada a objetos desde el comienzo, para evitar así un cambio de paradigmas de programación que ocurre en una aproximación tardía a la orientación a objetos.

Sin embargo, esta primera aproximación puede resultar problemática, porque se requiere de los estudiantes conocimientos simultáneos tales como:

- Conceptos generales de computación tales como el de código fuente, compilación, ejecución...
- Paradigmas independientes de programación como son la asignación, invocación y declaración de procedimientos y funciones, paso de parámetros...
- Conceptos específicos de la orientación a objetos, tales como clase, objeto, constructor, accesos...

2.1.4. Problemas con la visualización de la orientación a objetos

La dinámica de la ejecución de un programa orientado a objetos es bastante compleja cuando es enunciada completamente, pero estos detalles son esenciales para un correcto entendimiento de la programación orientada a objetos por parte del estudiante.

Enunciamos los principales escenarios que pueden ocurrir:

- **Creación de objetos**: la declaración de la clase es consultada para determinar los campos y así la memoria puede ser asignada a ese objeto.
  Después, bien por defecto o por inicializaciones explicitas de los campos, el constructor es ejecutado. Las instrucciones del constructor son de nuevo obtenidas de la declaración de la clase. Desde que los constructores son más frecuentemente usados para inicializar campos, el flujo de datos va desde los parámetros actuales a los parámetros formales y a continuación por asignación, hasta los campos del objeto.

- **Invocación de un método de un objeto**: la declaración de la clase es consultada para obtener la secuencia de instrucción del método. Los parámetros actuales son evaluados y pasados a parámetros formales. Las instrucciones son ejecutadas, donde se referencian identificadores son para identificar que son declarados en la clase, mientras los propios valores son campos del objeto.

Estos escenarios son esenciales para el entendimiento de la programación orientada a objetos y deben ser concebidos por el estudiante. En la terminología del constructivismo, un modelo mental debe ser construido. Estos detalles de conocimiento deben ser explícitamente enseñados, porque hay demasiado trabajo personal y los estudiantes son muy propensos a la construcción de modelos no viables.

De particular importancia además es la interacción entre las declaraciones estáticas de campos y métodos en las clases y el acceso dinámico y modificación de valores, en cada objeto de una clase.

Algunos de los errores que se han encontrado en la enseñanza temprana de la orientación a objetos son los siguientes: [Ben-Ari, Ragonis, Ben-Bassat, 2002]

- La diferencia entre una clase y el objeto de una clase.

- La ejecución de un constructor como parte de la creación de un objeto.
- Las operaciones solo pueden ser invocadas por objetos.
- El estado de un objeto (los valores de sus campos) y el hecho de que una operación puede cambiar un estado.
- La conexión entre ejecución de una operación y su código fuente.
- Parámetros actuales vs. Parámetros formales.
- La relación de usos entre clases, en particular, el valor de un campo de un objeto puede ser un objeto.

2.1.5. Herramientas de ayuda a la enseñanza de la orientación a objetos

Mientras un número de herramientas útiles han sido desarrolladas, existe un área considerable para la mejora. Específicamente, la introducción a la orientación a objetos en niveles inferiores al universitario tiene un desarrollo muy reciente que puede beneficiarse de más apoyo de este tipo de herramientas.

Estas herramientas de apoyo se basan en la animación para ayudar al estudiante a visualizar lo que un programa orientado a objetos está realizando en todo momento.

Los principios de diseño de animación deben corresponderse con completitud y continuidad. Completitud significa que cada característica del programa debe ser visualizada. Y continuidad quiere decir, que la animación debe establecer las relaciones entre el programa explícito y las acciones. Por ejemplo, Jeliot 3 muestra como los valores de subexpresiones de una expresión contribuyen para dar ese valor. Esto significa que los objetos visuales que representan subexpresiones deben permanecer visibles hasta que todos ellos hayan sido evaluados. Entonces, esos objetos son animados para formar la expresión.

El movimiento de enseñanza temprana de orientación a objetos (“objects-early movement”) ha estado argumentando, desde hace tiempo, que es importante enseñar adecuadamente unas buenas prácticas de orientación a objetos desde el comienzo, para evitar tener que corregir u eliminar las malas prácticas y para ello en muchas ocasiones es necesario apoyarse en herramientas que faciliten tal entendimiento de la programación orientada a objetos. [Henriksen, Kölling, 2004]

2.2.Entornos de visualización

El propósito de este apartado es revisar una serie de herramientas que ayudan a visualizar los objetos y su interacción de forma que se puedan entender mejor los conceptos abstractos de la programación orientada a objetos.

La elección de las herramientas analizadas se ha basado en la búsqueda de aplicaciones desarrolladas o en proceso de desarrollo que mantuviesen objetivos afines a la problemática planteada en el presente proyecto.

2.2.1 BlueJ

BlueJ es un entorno de desarrollo integrado para el lenguaje de programación Java, desarrollado principalmente para la educación, pero también adecuado para los pequeños desarrollos de software.

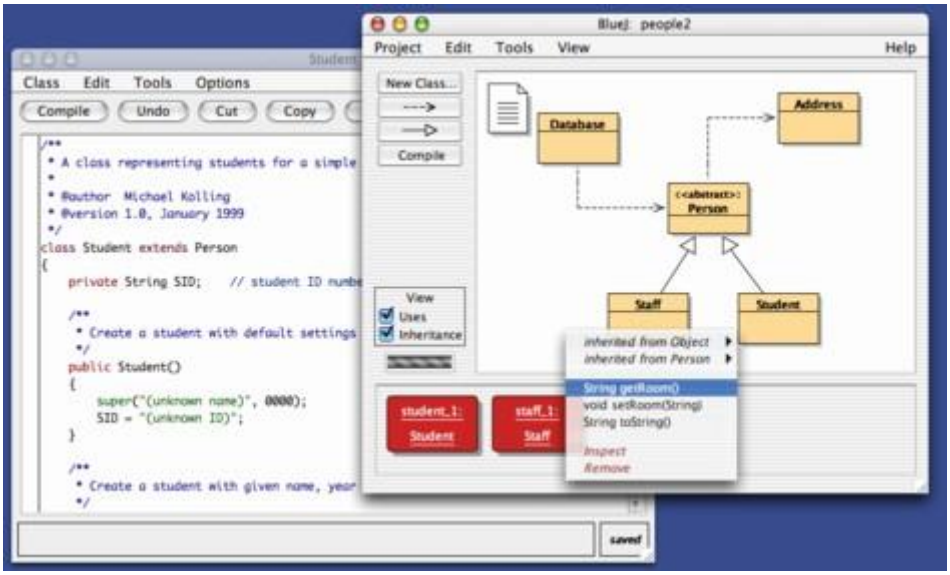


Figura 1. Programa BlueJ

Fue desarrollado para apoyar el aprendizaje y la enseñanza de la programación orientada a objetos, y su diseño se diferencia de otros entornos de desarrollo como consecuencia de ello. La pantalla principal muestra gráficamente la estructura de clases de la aplicación que se esté desarrollando (con un diagrama UML), y se

puede interactuar con los objetos. Esta interacción, junto con un limpio y sencillo interfaz de usuario, permite una fácil experimentación con objetos en desarrollo. Conceptos orientados a objetos (clases, objetos, la comunicación a través de llamadas a los métodos) están representadas visualmente. [1]

Esta herramienta está más encaminada a una introducción a la programación a nivel universitario. Se asume que el estudiante está interesado en aprender programación y que se quiere hacer de una manera organizada.

2.2.2. Jeliot 3

Jeliot 3 es una aplicación que permite visualizar cómo un programa en Java es interpretado. Muestra el funcionamiento en una pantalla como una animación continúa, que permite al estudiante seguir paso a paso la creación de variables, las llamadas a los métodos, etc. [2]

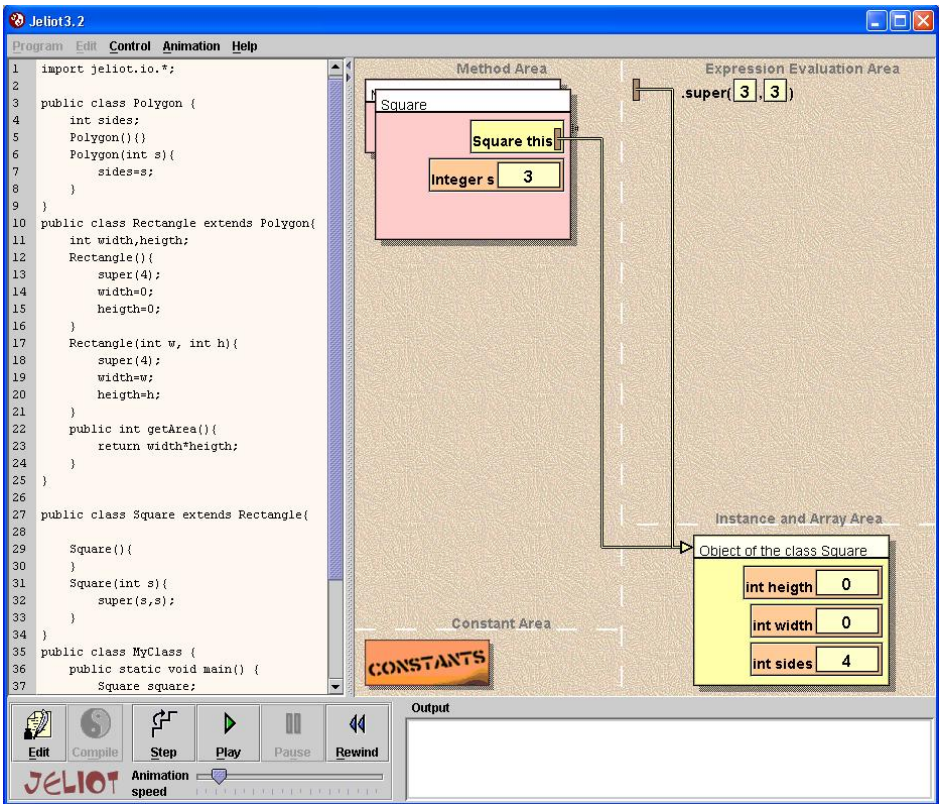


Figura 2. Programa Jeliot 3

Se ejecuta en prácticamente cualquier plataforma, incluyendo Windows, Linux, Mac. El único requisito es tener un JRE instalado en su sistema.

Hay estudios que demuestran, que proporciona a los que se inician en la programación orientada a objetos, modelos mentales y el vocabulario necesario para describir la ejecución del programa. [Ben-Bassat Levy et al., 2003]

El método recomendado para ser utilizado en clases de iniciación a la programación orientada a objetos, es visualizar los conceptos de programación durante las clases para proporcionar a los estudiantes un modelo concreto de la ejecución del programa. Los estudiantes pueden usar la herramienta en su tiempo libre para visualizar sus propios programas, aunque no tiene por objeto el desarrollo de programas.

2.2.3. Alice

Alice es un innovador entorno de programación 3D que facilita la creación de una animación para contar una historia, jugar a un juego interactivo, o un video para compartir en la web.

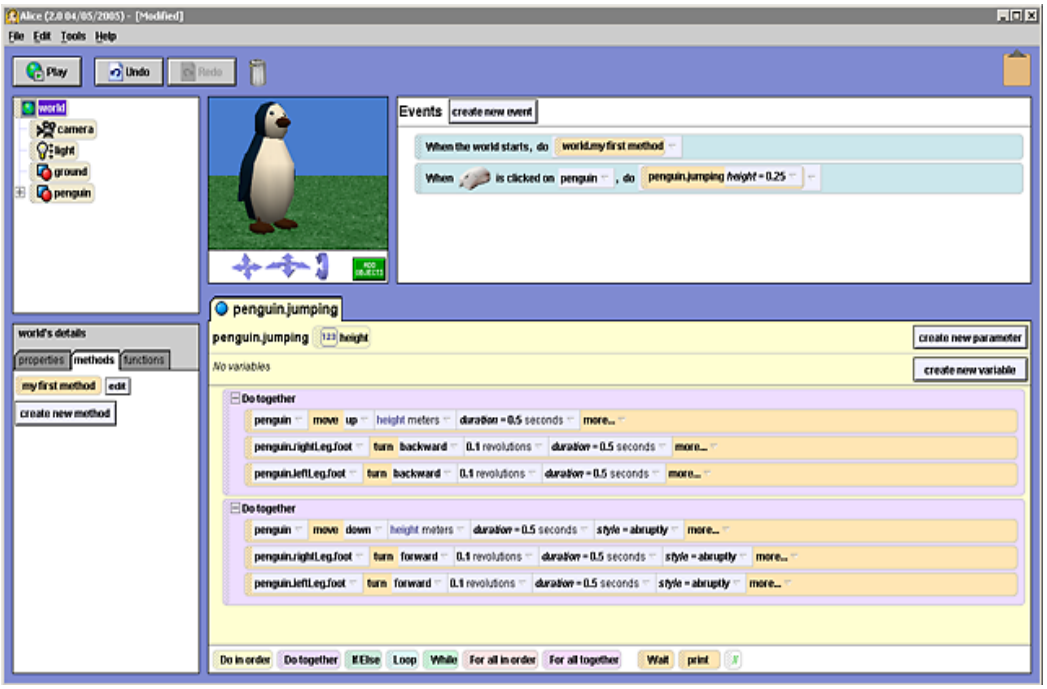


Figura 3. Programa Alice

Es una herramienta de libre acceso de enseñanza diseñada para estudiantes que se exponen por primera vez a la programación orientada a objetos. Permite a los estudiantes aprender los conceptos fundamentales de programación en el contexto de la creación de películas animadas y sencillos videojuegos.

El programa se desarrolló prioritariamente para solucionar tres problemas fundamentales del software educativo: [3]

1. La mayoría de los lenguajes de programación están diseñados para producir otros programas, cada vez más complejos. Alice está diseñado únicamente para enseñar a programar.
2. Está íntimamente unido a su IDE (Integrated Development Environment, Entorno de Desarrollo Integrado). No hay que recordar ninguna sintaxis especial. Acepta tanto el modelo de programación orientada a objetos como la dirigida a eventos.
3. Alice está diseñada para el público que normalmente no se enfrenta a problemas de programación, tales como alumnos de secundaria, mediante un sistema de «arrastrar y soltar».

Permite a los estudiantes ver cómo ejecutar sus programas de animación, lo que les ayuda a comprender fácilmente la relación entre la programación de las declaraciones y el comportamiento de los objetos en su animación. Mediante la manipulación de los objetos en su mundo virtual, los estudiantes adquieren experiencia con todas las estructuras de la programación general que se enseñan en un curso introductorio de programación.

Se han desarrollado materiales de instrucción para apoyar a los estudiantes y profesores en la utilización de este nuevo enfoque. Los recursos incluyen libros de texto, la experiencia, muestras de los programas, bancos de pruebas, y más.

2.2.4. Greenfoot

El sistema Greenfoot es una combinación de framework y entorno para la creación de aplicaciones de simulación interactivas en un plano bidimensional en el lenguaje de programación Java. Es adecuado para nuevos programadores.

En Greenfoot la visualización de objetos y la interacción entre ellos son los elementos clave.

Una manera de visualización del sistema es como una extensión del banco de objetos de BlueJ. Greenfoot extiende la idea del banco de objetos al mundo de los objetos. En este mundo los objetos tienen una apariencia gráfica y una posición en el mundo. La interacción directa con estos objetos es todavía posible, como en el original BlueJ pero el comportamiento de los objetos puede ser ahora observado directamente al ver los cambios en la posición y apariencia individual de los objetos. [4]

El propio mundo de los objetos (visible como área subyacente detrás de los objetos Greenfoot) llega a ser un objeto programable, interactivo integrado en la estructura de la aplicación.

Para obtener información más detallada sobre esta herramienta, leer el Anexo que se incluye al final del documento.

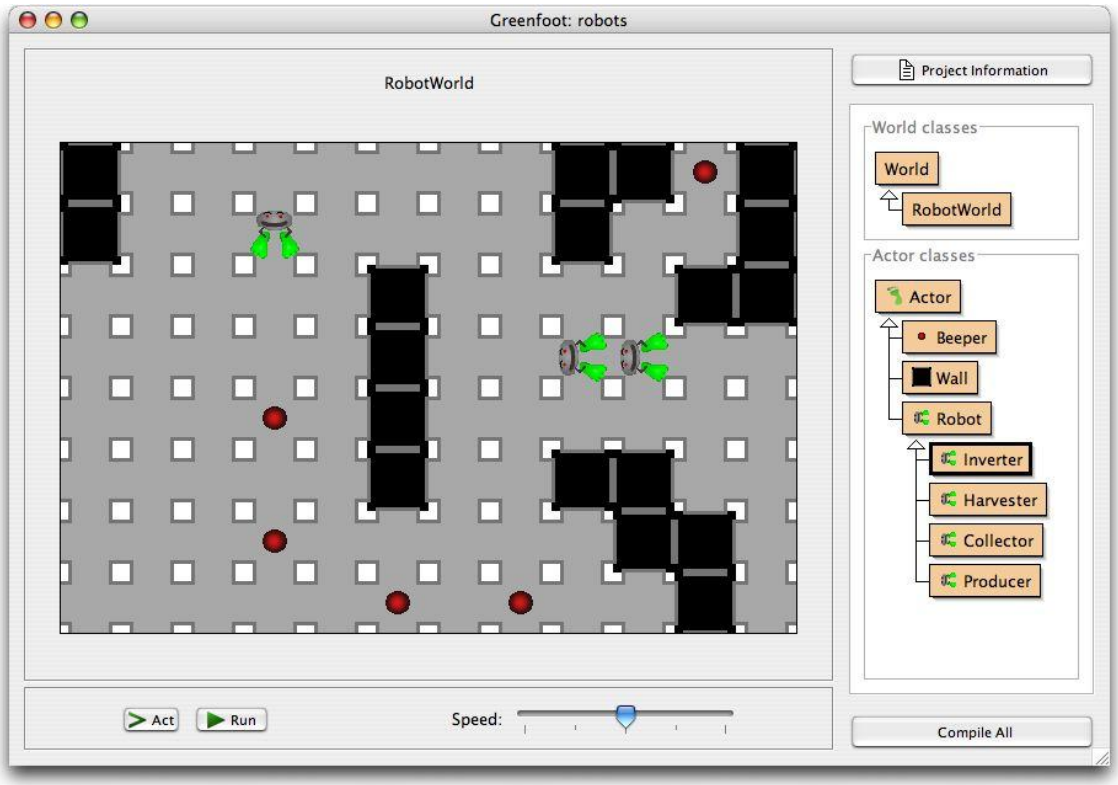


Figura 4. Programa Greenfoot

2.3. Análisis de las herramientas de visualización

En este apartado se estudiarán las ventajas e inconvenientes de las herramientas, así como la manera en que dichas herramientas afrontan la problemática planteada.

Todas las herramientas mencionadas anteriormente presentan un especial énfasis en la visualización, por lo que todas ellas cubren la problemática del aprendizaje de algunos conceptos de programación orientada a objetos mediante la visualización de los objetos. Sin embargo no todas ofrecen las mismas posibilidades ni todas son igual de intuitivas, es por ello que se comentarán a continuación las principales ventajas e inconvenientes de las herramientas, así como las razones de la herramienta seleccionada como la más adecuada para el desarrollo del proyecto.

2.3.1. BlueJ

La principal ventaja que presenta esta y todas las demás herramientas es el énfasis en la visualización. Sin embargo dentro de esta característica existen numerosos aspectos que son las que diferencian una herramienta de otra.

El entorno de desarrollo integrado, junto con el editor "built-in" (compilador, máquina virtual, depurador) ayudan a los usuarios a afianzar pequeños conceptos generales de computación tales como código fuente, compilación ejecución, etc. una de las principales problemáticas que existen en una primera aproximación a la orientación a objetos.

Además posee una estructura gráfica de clases para que los usuarios creen sus propios diagramas de clases. Esto les permitirá adquirir y poner en prácticas conceptos y estructuras de la programación orientada a objetos con los diagramas de clase UML (Lenguaje Unificado de Modelado). Sin embargo desde mi punto de vista es necesario que conceptos de UML sean enseñados de manera independiente, tratando así no confundir a los alumnos.

En cuanto a la visualización, BlueJ dispone de un editor textual y gráfico, con una interfaz de usuario fácil de usar y adecuada para principiantes, lo que permite la creación interactiva de objetos y la invocación de los métodos de esos objetos. Esta creación interactiva y visual de los objetos permite a los usuarios esclarecer la diferencia entre clase y objeto. Sin embargo existe una

deficiencia en cuanto a esta representación visual de los objetos en BlueJ. Proporciona únicamente el nombre y la clase del objeto, en ningún momento se muestra ninguna pista de cuál es el estado del objeto o el comportamiento. BlueJ proporciona interacción directa con los objetos pero no proporciona visualización directa del estado o comportamiento del objeto. [Henriksen, Kölling, 2004]

Estudios realizados mediante encuestas a alumnos se consideran los puntos débiles de esta herramienta la estabilidad del producto y la dificultad en la instalación.

Además los propios autores de BlueJ detectan varias áreas con problemas potenciales: BlueJ está planteado como un entorno para programadores principiantes, pero es interesante que los estudiantes a medida que vayan cogiendo experiencia se vayan familiarizando con otros entornos más profesionales o entornos que le permitan realizar todo tipo de programas. Por último, según el planteamiento pedagógico de los autores de BlueJ hay peligro de que se dedique mucho tiempo a conceptos de orientación a objetos y se descuiden otros conceptos como las estructuras de datos y algoritmos; hay que tener en cuenta que estos conceptos siguen siendo importantes y dedicarles tiempo para que los estudiantes también los entiendan y puedan aplicarlos correctamente.

2.3.2. Jeliot3

Es una herramienta adecuada para la iniciación a la programación pues permite una mejor comprensión de la creación de objetos, variables e invocación de métodos mediante una animación contínua o paso a paso.

Esta herramienta hace especial énfasis en la visualización de la creación de objetos con los constructores, la herencia y el control de estado de los objetos, lo que proporciona algunos conceptos de la programación orientada a objetos, pero quizá no los suficientes.

Jeliot3 es considerada desde mi punto de vista, una herramienta adecuada para la comprensión de la sintaxis java, la creación de variables, la asignación de valores, el uso de operadores, así como el control de flujo. Pero no para aprender los conceptos básicos de la programación orientada a objetos, así como la diferencia entre una clase y un objeto, la invocación de los métodos y la interacción con los objetos.

Otro de los problemas que presenta esta herramienta es el límite en el tamaño de los programa, no existe tal límite, todas las clases están en un archivo y debido al espacio limitado del fotograma de animación, la visualización de muchos objetos no será posible.

Una de las mayores deficiencias de esta herramienta es la interacción con los objetos, pues una vez compilado el código y ejecutado, el usuario no ejerce ningún tipo de interacción, únicamente observa el resultado e intenta comprender, lo que hace que el uso de la herramienta sea menos llamativo para los usuarios.

Con todo esto, hace llegar a la conclusión de que se trata de una herramienta que permite más una comprensión de la sintaxis Java, (la creación de variables, el uso de arrays, etc.) que la problemática planteada, los conceptos de programación orientada a objetos (diferencia entre clase y objeto, invocación a métodos, interacción con los objetos, estado de los objetos, etc.).

2.3.3. Alice

Al igual que las herramientas estudiadas anteriormente Alice permite a los estudiantes comprender mejor los conceptos de la orientación a objetos, así como centrarse en las ideas de los proyectos, y no tanto en la programación.

Es fácil de usar y existen numerosos tutoriales y libros para aprender a utilizar la herramienta correctamente, lo que hace que se trate de una herramienta fácil de usar como apoyo para el aprendizaje para la programación orientada a objetos.

Destacar además la gran variedad de interacción con los objetos. El usuario puede: [Cooper, Dann, Pausch, 2003]

- Arrastrar y soltar un objeto y hacer otros ajustes necesarios con el ratón.
- Fijar sus posiciones y hacer los ajustes necesarios manualmente de manera interactiva, así como la invocación de métodos sobre los objetos.
- Se puede escribir código para configurar puntos de vista de los objetos después de que el programa comienza a correr, pero antes de que el empiece la animación.

Son estas interacciones con los objetos las que permiten comprender mejor la diferencia entre clase y objeto, la invocación a métodos de los objetos y el estado de los mismos.

No obstante, uno de los problemas de esta herramienta es precisamente en esta interacción con los objetos. Los objetos son creados y añadidos al mundo 3D. Estos objetos tienen una serie de propiedades que pueden ser inspeccionadas y cambiadas antes de que empiece la animación, sin embargo, una vez iniciada la animación no es posible inspeccionar el estado de los objetos ni interactuar con ellos. [Henriksen, 2004] No es un inconveniente de gran envergadura pero sí marca la diferencia en el momento de selección de la herramienta más adecuada.

Por último destacar el problema con las animaciones. Existe una gran diferencia entre las animaciones básicas y genéricas que proporciona la herramienta Alice y las animaciones que se pueden pretender crear. Para alumnos que se encuentran en una primera aproximación a Java es fácil entender las animaciones básicas ofrecidas por Alice, sin embargo, a medida que los alumnos avanzan en conocimientos, el interés aumenta y quieren crear animaciones de mayor complejidad, lo cual es demasiado complicado y reduce su interés. Un ejemplo es el querer crear una animación en la que un personaje pueda andar. Hace que el usuario deba combinar al menos ocho animaciones diferentes en las que mover las piernas, las rodillas, los pies, las extremidades superiores, etc., para poder moverlo. Es por ello que los estudiantes valoraron la cantidad de líneas de código necesarias para ello y decidieron reducir sus objetivos. [Kelleher, 2006]

2.3.4. Greenfoot

Esta herramienta tal y como se explicó en su descripción en el apartado anterior es adecuada para nuevos programadores, debido a que proporciona tanto un editor como una parte visual que permite interaccionar con los objetos.

Estos son precisamente sus elementos clave, la visualización de los objetos y la interacción. Greenfoot extiende la idea del banco de objetos que proporcionaba BlueJ al mundo de los objetos permitiendo una interacción directa con ellos. Por tanto esta herramienta proporciona todos los beneficios de BlueJ pero el comportamiento de los objetos puede ser ahora observado directamente al ver los cambios en la posición y apariencia individual de los objetos. Proporciona además una clara diferenciación entre clase y objeto.

Además de los conceptos de diferenciación entre clase y objeto, la invocación a métodos y el acceso al estado y comportamiento de los objetos, al igual que en BlueJ, el entorno de desarrollo integrado, junto con el editor "built-in" ayuda a los usuarios a afianzar pequeños conceptos generales de computación tales como código fuente, compilación ejecución, etc. [Henriksen, 2004]

Para permitir la visualización de los objetos en el mundo de Greenfoot es necesario el uso de un api que proporciona la herramienta, la cual es necesario conocer en profundidad para la realización de cualquier desarrollo en el entorno.

Destacar además de que se trata de una herramienta que actualmente se encuentra en desarrollo, por lo que se van subsanando aquellos errores que los usuarios van encontrando y sugiriendo a través de un foro del que dispone la página web http://www.greenfot.org. Además en la actualidad está siendo desarrollado un libro tutorial llamado "Introduction to Programming with Greenfoot, Object-Oriented Programming in Java with Games and Simulations", que facilitará el uso de la herramienta.

3. Gestión del proyecto

Este capítulo del documento se centra en estudiar las necesidades planteadas, tratando las posibles restricciones que puedan condicionar el estudio. Como complemento indispensable a este análisis, se realizarán las correspondientes estimaciones en tiempo, recursos y coste.

3.1. Estimación de tiempo y recursos

La estimación del esfuerzo que lleva el proyecto actual se ha realizado usando la herramienta COCOMO II [5], una herramienta de estimación basada en ecuaciones matemáticas. COCOMO II está pensado para la estimación del tamaño de proyectos pequeños y de tamaño medio por lo que se ajusta perfectamente al proyecto actual y permite obtener unos datos fiables a partir de los factores analizados en los apartados siguientes.

- **Modelo de estimación:**

Debido al punto actual en el que se encuentra el proyecto, se ha establecido como modelo de estimación el modelo Early-Design (Diseño preliminar), utilizado en las fases más tempranas de un proyecto software.

- **Factores de escala:**

Afectan de forma global al proyecto en desarrollo, para lo que se han analizado los precedentes en el área de negocio (*Precedentedness*), la flexibilidad del desarrollo (*Development Flexibility*), los riesgos de la arquitectura (*Architecture / risk resolution*), la cohesión del equipo de desarrollo (*Team cohesion*) y la madurez del proceso (*Process maturity*). [6] Véase Figura 5

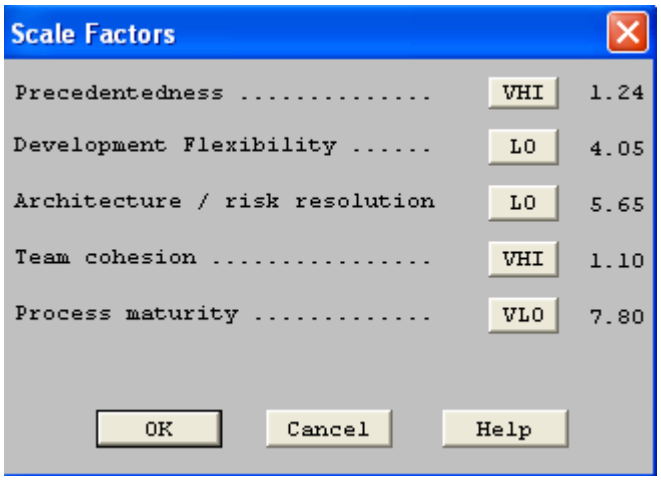


Figura 5. Factores de escala

- **Multiplicadores de esfuerzo:**

Estos factores ajustan el esfuerzo estimado del producto en desarrollo, para lo que se han analizado los atributos del producto, de plataforma, de personal y de proyecto. [6] Véase Figura 6

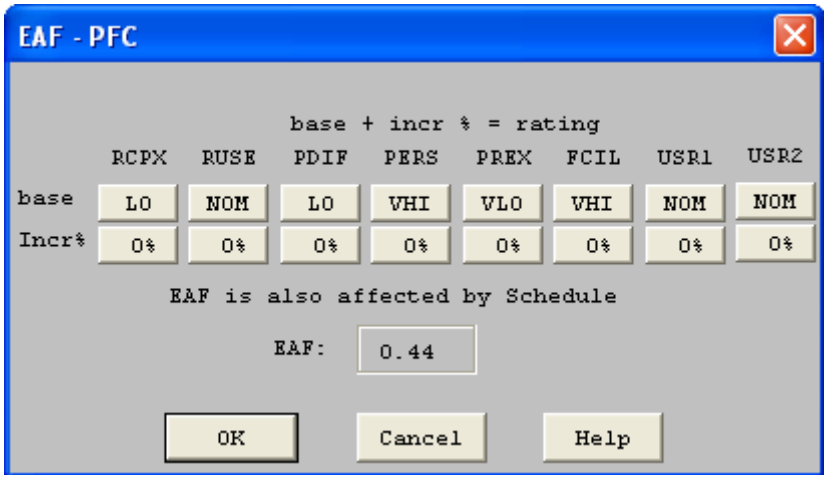


Figura 6. Multiplicadores de esfuerzo

- **Tamaño del proyecto:**

Para la estimación del tamaño de proyecto se han escogido el número de líneas de código fuente (LOC) del prototipo que previamente se ha desarrollado. A este número, que define una medida física de líneas, se ha aplicado una reducción del 30% [Referencia] para obtener un valor lógico que mejore la estimación del tamaño. [7]

Language	To Derive Logical SLOC
Assembly and Fortran	Assume Physical SLOC = Logical SLOC
Third-Generation Languages (C, Cobol, Pascal, Ada 83)	Reduce Physical SLOC by 25%
Fourth-Generation Languages (SQL, Perl, Oracle)	Reduce Physical SLOC by 40%
Object-oriented Languages (Ada 95, C++, Java, Python)	Reduce Physical SLOC by 30%

Tabla 1. Transformación líneas físicas a líneas lógicas de código según el lenguaje

- **Coste medio del personal:**

Este factor se ha calculado en función de la media del precio por hora y las horas por mes para personal con poca experiencia en el sector.

Partiendo de los factores definidos en la herramienta COCOMO II, realiza una estimación del esfuerzo, tiempo y costes. (Figura 7)

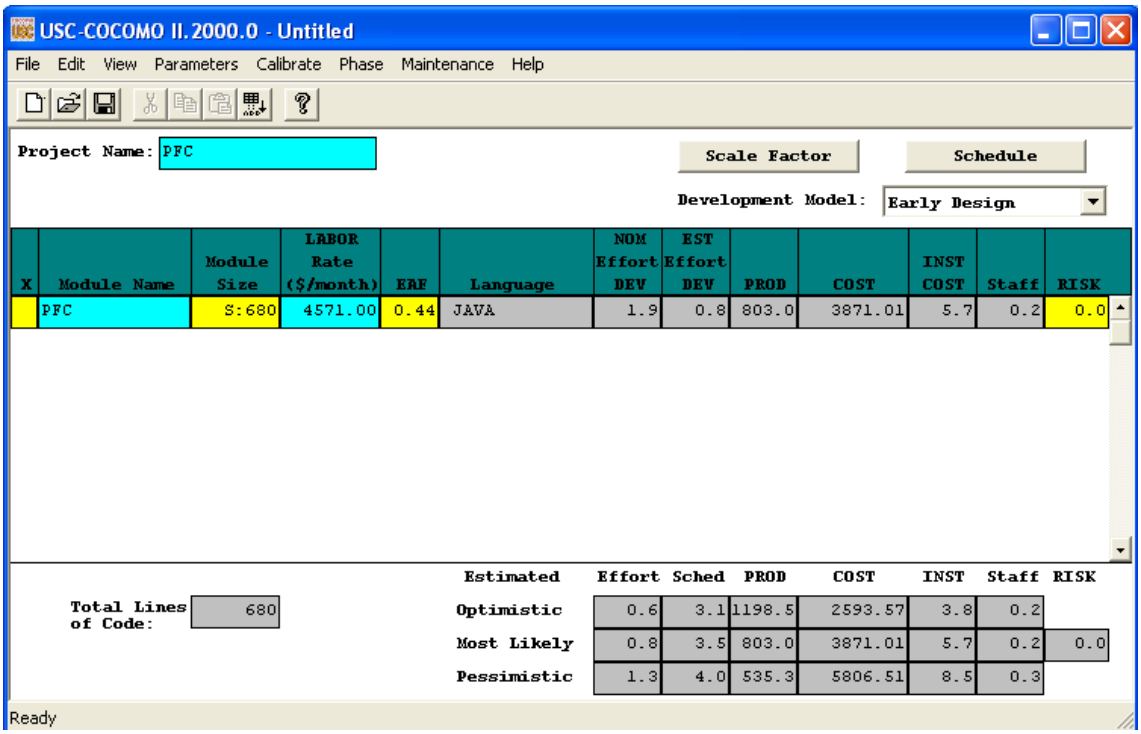


Figura 7. Estimación COCOMO II

Tal y como se observan en los resultado mostrados por COCOMO II, se estima una duración de 3.5 meses con un personal empleado de 0.2 personas. Estos datos son los que COCOMO II estima como óptimos para el proyecto actual, sin embargo como es evidente estos datos pueden variar por diversas condiciones, bien sea por cuestiones de recursos, de tiempo, etc.

3.2.Gestión de recursos

Se recoge en este apartado el diagrama RBS (*Resource Breakdown Structure*) (Figura 8), diagrama jerárquico que representan los recursos humanos y materiales del proyecto. Los principales objetivos son: mostrar gráficamente la organización humana del proyecto y reflejar la estructura de recursos materiales necesarios para la realización del proyecto.

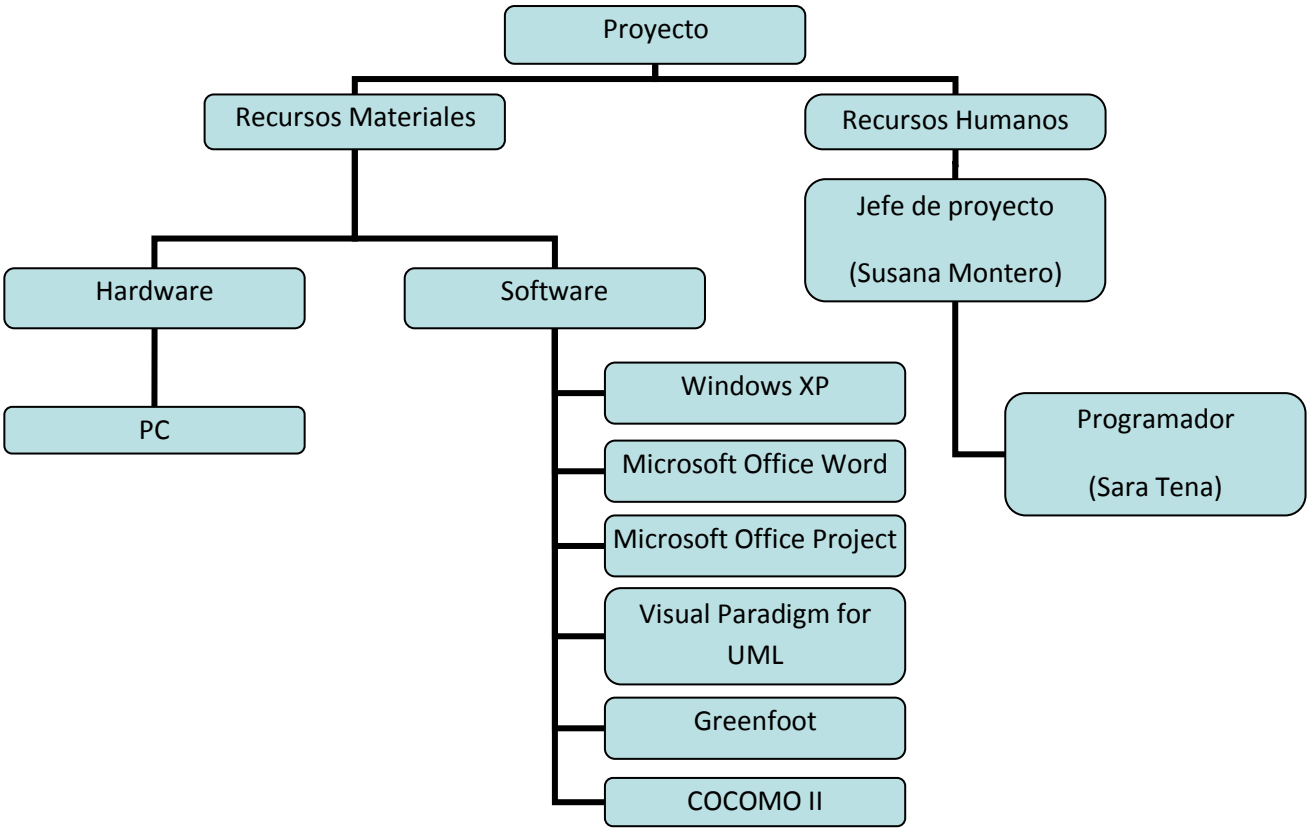


Figura 8. Gestión de recursos humanos y materiales

A continuación se enumeran cada uno de los roles del equipo de proyecto así como la explicación general de su cometido:

- **Jefe de proyecto:** Responsable del proyecto. Se encarga de la dirección para la realización del mismo. Es la persona que mantiene contacto con el usuario final.
- **Programador:** Responsable del análisis y diseño del proyecto, implementación y pruebas.

3.3. Plan de trabajo

En este apartado se mostrarán las tareas que han llevado a cabo por el desarrollo completo del proyecto.

3.3.1. Identificación de tareas

1. **Estudio preliminar:** Esta tarea está compuesta por dos subtareas:
    a. **Estado de la cuestión:** Se realiza un estudio preliminar de la problemática y las posibles herramientas.
    b. **Esbozo de la solución:** Se plantea una solución en rasgos generales.
2. **Desarrollo:**
    a. **Análisis de requisitos:** Se analizan las necesidades de los usuarios.
    b. **Diseño:** Se estudian las distintas maneras de desarrollar la aplicación.
    c. **Implementación:** Se codifica la solución.
    d. **Pruebas:** Se realizan pruebas para comprobar el correcto funcionamiento de la solución.
3. **Documentación:** Se divide en diversas tareas:
    a. **Introducción:** Presenta el trabajo realizado y el contenido del documento.
    b. **Estado de la cuestión:** Se presenta el contexto de realización del trabajo realizando una revisión de las herramientas. En mayor profundidad que el estado de la cuestión realizado en la tarea de 'Estudio preliminar'.
    c. **Gestión del proyecto:** Se muestra la planificación del proyecto, así como los recursos necesarios para su realización.

d. **Planteamiento y solución:** Muestra un pequeño resumen de la problemática planteada, junto con los requisitos del usuario y la solución llevada a cabo.

e. **Análisis:** Se analiza y especifican los requisitos y restricciones del sistema desde el punto de vista de la funcionalidad.

f. **Diseño:** Comprende el desarrollo de un modelo orientado a objetos de un sistema software para implementar los requerimientos identificados.

g. **Implementación:** Recoge los problemas y decisiones llevadas a cabo en el proceso de implementación.

h. **Pruebas:** Se comprueba si el resultado se corresponde con la especificación del sistema. Tiene como objetivo mostrar la consecución de las funcionalidades planteadas.

i. **Conclusión:** Se expone la opinión personal sobre el trabajo realizado así como los posibles trabajos futuros.

j. **Bibliografía:** Recoge las referencias utilizadas para la realización del presente proyecto.

k. **Anexos:** Contiene aquellos anexos considerados necesarios para el proyecto.

3.3.2. Diagrama Gantt

El diagrama de Gantt es una popular herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

En este caso se mostrará un diagrama de Gantt del tiempo planificado (Figura 9 y Figura 10), que mostrará lo que sería inicialmente el tiempo del proyecto, y otro diagrama de Gantt en el que se muestra el tiempo real (Figura 11 y Figura 12) que ha llevado el desarrollo del proyecto.

Diagrama de Gantt planificado

Se muestra en la siguiente imagen (Figura 9) el tiempo en días planificado para cada una de las tareas descritas anteriormente:

	Nombre de tarea	Duración	Comienzo	Fin	Predeces	Nombres de los recursos
1	Inicio del proyecto	0 días	mié 01/10/08	mié 01/10/08		
2	**Estudio preliminar**	**40 días**	**jue 02/10/08**	**mié 26/11/08**		
3	Estado de la cuestión	30 días	jue 02/10/08	mié 12/11/08	1	Programador[50%]
4	Esbozo de la solución	10 días	jue 13/11/08	mié 26/11/08	3	Programador[25%]
5	**Desarrollo**	**34 días**	**jue 27/11/08**	**mar 13/01/09**		
6	Análisis de requisitos	7 días	jue 27/11/08	vie 05/12/08	4	Programador[25%]
7	Diseño	2 días	lun 08/12/08	mar 09/12/08	6	Programador[25%]
8	Implementación	20 días	mié 10/12/08	mar 06/01/09	7	Programador[25%]
9	Pruebas	5 días	mié 07/01/09	mar 13/01/09	8	Programador[25%]
10	**Documentación**	**59 días**	**mié 14/01/09**	**lun 06/04/09**		
11	Introducción	7 días	mié 14/01/09	jue 22/01/09	9	Programador[50%]
12	Estado de la cuestión	7 días	vie 23/01/09	lun 02/02/09	11	Programador[35%]
13	Gestión de proyecto	3 días	mar 03/02/09	jue 05/02/09	12;20;21	Programador[50%]
14	Planteamiento y solución	4 días	vie 06/02/09	mié 11/02/09	13	Programador[50%]
15	Análisis	7 días	jue 12/02/09	vie 20/02/09	14	Programador[50%]
16	Diseño	7 días	lun 23/02/09	mar 03/03/09	15	Programador[50%]
17	Implementación	7 días	mié 04/03/09	jue 12/03/09	16	Programador[50%]
18	Pruebas	12 días	vie 13/03/09	lun 30/03/09	17	Programador[50%]
19	Conclusión	5 días	mar 31/03/09	lun 06/04/09	18	Programador[50%]
20	Bibliografía	1 día	vie 23/01/09	vie 23/01/09	11	Programador[5%]
21	Anexos	7 días	vie 23/01/09	lun 02/02/09	11	Programador[10%]
22	Fin del proyecto	0 días	lun 06/04/09	lun 06/04/09	19	

Figura 9. Tareas del proyecto en tiempo planificado

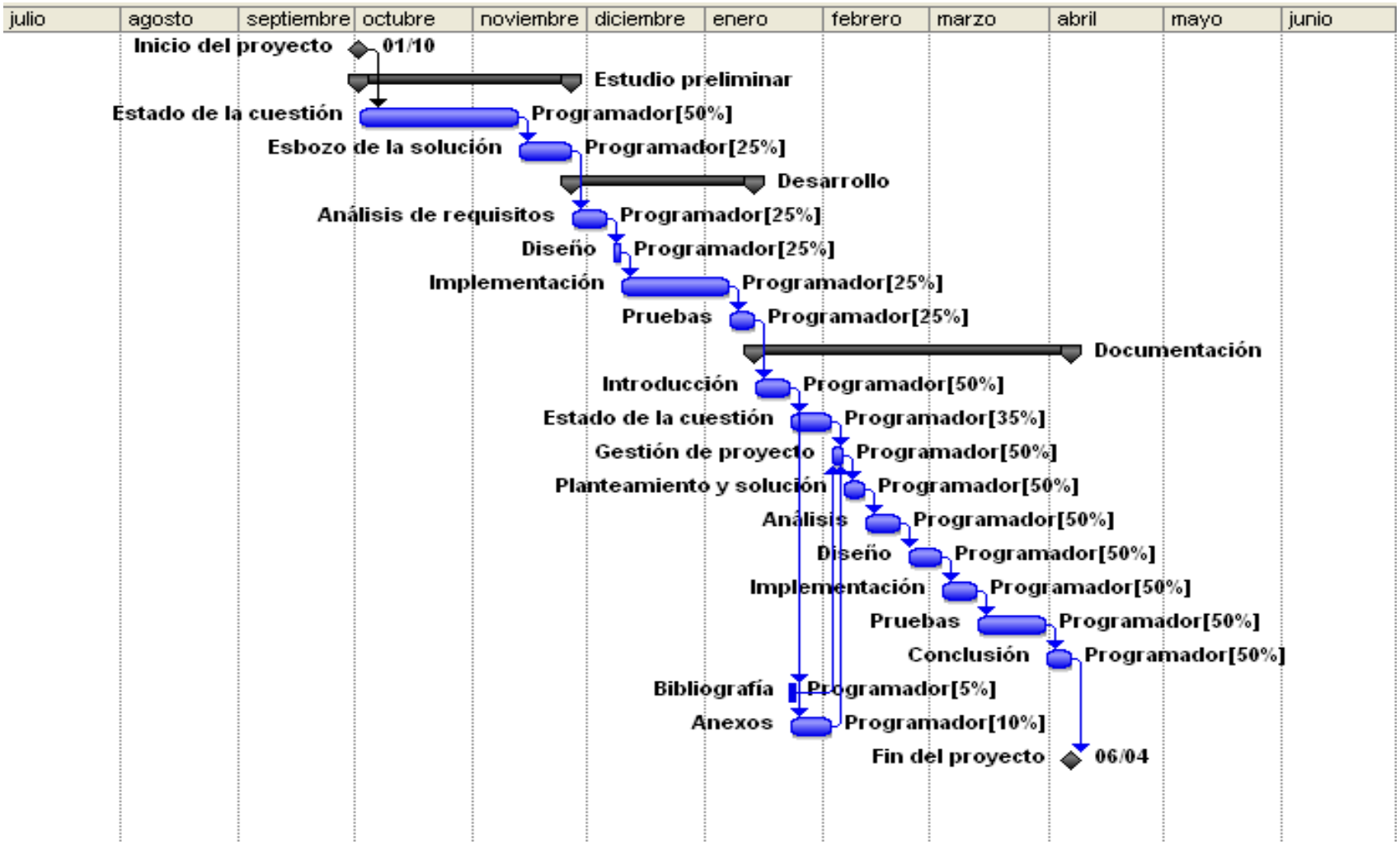


Figura 10. Diagrama de Gantt planificado

Como se observa en el diagrama de Gantt planificado (Figura 10), la duración total del proyecto son 133 días (4,4 meses aproximadamente), teniendo su inicio el día 01 de Octubre de 2008 y finalizando el día 06 de Abril de 2009.

Sin embargo en ocasiones no es posible seguir la planificación planteada en un inicio, debido a que pueden surgir imprevistos y complicaciones, o problemas con los que no esperábamos encontrarnos y que suponen un retraso en la planificación y en consecuencia tener que re-planificar.

Diagrama de Gantt real

Se muestra en la siguiente imagen (Figura 11) el tiempo en días que han sido necesarios para llevar a cabo cada una de las tareas descritas anteriormente:

	Nombre de tarea	Duración	Comienzo	Fin	Predeces	Nombres de los recursos
1	Inicio del proyecto	0 días	mié 01/10/08	mié 01/10/08		
2	**Estudio preliminar**	**60 días**	**jue 02/10/08**	**mié 24/12/08**		
3	Estado de la cuestión	30 días	jue 02/10/08	mié 12/11/08	1	Programador[50%]
4	Esbozo de la solución	30 días	jue 13/11/08	mié 24/12/08	3	Programador[25%]
5	**Desarrollo**	**34 días**	**jue 09/04/09**	**mar 26/05/09**		
6	Análisis de requisitos	7 días	jue 09/04/09	vie 17/04/09		Programador[25%]
7	Diseño	2 días	lun 20/04/09	mar 21/04/09	6	Programador[25%]
8	Implementación	20 días	mié 22/04/09	mar 19/05/09	7	Programador[25%]
9	Pruebas	5 días	mié 20/05/09	mar 26/05/09	8	Programador[25%]
10	**Documentación**	**74 días**	**mié 27/05/09**	**lun 07/09/09**		
11	Introducción	7 días	mié 27/05/09	jue 04/06/09	9	Programador[50%]
12	Estado de la cuestión	15 días	vie 05/06/09	jue 25/06/09	11	Programador[35%]
13	Gestión de proyecto	3 días	vie 26/06/09	mar 30/06/09	12;20;21	Programador[50%]
14	Planteamiento y solución	7 días	mié 01/07/09	jue 09/07/09	13	Programador[50%]
15	Análisis	10 días	vie 10/07/09	jue 23/07/09	14	Programador[50%]
16	Diseño	7 días	vie 24/07/09	lun 03/08/09	15	Programador[50%]
17	Implementación	10 días	mar 04/08/09	lun 17/08/09	16	Programador[50%]
18	Pruebas	12 días	mar 18/08/09	mié 02/09/09	17	Programador[50%]
19	Conclusión	3 días	jue 03/09/09	lun 07/09/09	18	Programador[50%]
20	Bibliografía	2 días	vie 05/06/09	lun 08/06/09	11	Programador[5%]
21	Anexos	7 días	vie 05/06/09	lun 15/06/09	11	Programador[10%]
22	Fin del proyecto	0 días	lun 07/09/09	lun 07/09/09	19	

Figura 11. Tareas del proyecto

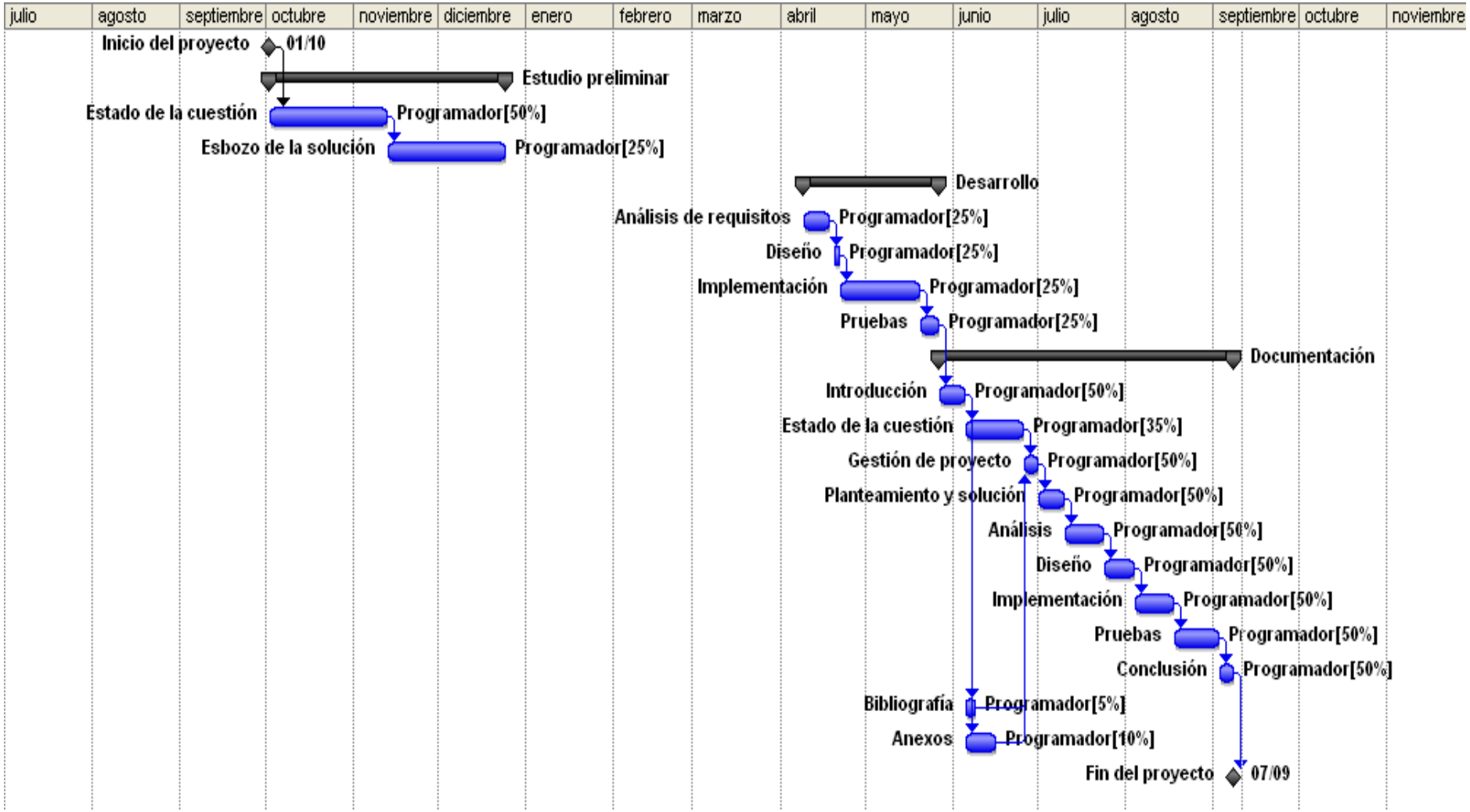


Figura 12. Diagrama Gantt real

En el diagrama de Gantt real (Figura 12), la duración total del proyecto son 168 días (5,4 meses aproximadamente). Se inicia el día 01 de Octubre de 2008 y finaliza el día 07 de Septiembre de 2009 debido a que ha existido un tiempo de parón en el desarrollo.

Si comparamos el Gantt planificado con el real podemos ver que existe un retraso de 35 días, debido a los problemas encontrados durante el desarrollo lo que incrementó el número de líneas de código (SLOC) calculadas previamente.

En ambos casos el rol de Analista y Programador es desempeñado por la misma persona, por lo que no es posible la realización de muchas tareas en paralelo. Únicamente tareas en la realización de la documentación, tareas como la Bibliografía y los Anexos, pueden ser realizadas en paralelo con otras tareas debido al menor esfuerzo que requieren.

Si comparamos los datos obtenidos con la estimación realizada con COCOMO II con el tiempo real del proyecto, según COCOMO la duración del proyecto debería de ser de 3.5 meses, sin embargo existe una diferencia aproximada de 1.9 meses.

3.4.Presupuesto

En este apartado se presenta el presupuesto inicial por el desarrollo del proyecto descrito en este documento: salarios de los trabajadores, equipos informáticos, material fungible y otros costes asociados con el desarrollo del mismo. A continuación se detalla el origen de cada coste:

Coste del personal

Para el cálculo del presupuesto del personal, se ha tomado el precio h/mes de una empresa de soluciones tecnológicas que proporciona desarrollo de software para los sectores de las telecomunicaciones, transporte, medioambiente y automatización industrial, entre otros. De la misma empresa se ha tomado el convenio de trabajo en el que se realiza una media diaria de 8,41 horas diarias, siendo un total de 1801,2 horas al año.

Con el precio de la hora al mes y las horas estimadas (532 h totales), se ha realizado el cálculo del coste del personal.

COSTE DEL PERSONAL		
Rol	**Precio de hora trabajada**	
Jefe de proyecto	24 €/hora (42000 €, convenio de 1801,2 horas)	
Programador	13 €/hora (22000 €, convenio de 1801,2 horas)	
Rol	**Horas estimadas de trabajo**	**Precio total por horas trabajadas**
Jefe de proyecto	266 (promedio de 165,5 h/mes)	6384 €
Programador	266 (promedio de 165,5 h/mes)	3458 €
	TOTAL:	**9842 €**

Tabla 2. Coste del personal

Coste de equipos informáticos

COSTE DE EQUIPOS INFORMÁTICOS		
Concepto	**Cantidad**	**Precio por unidad**
HP PAVILION P6010ES-M. Cpu + Monitor 20"	2	499 €
	TOTAL:	998€

Tabla 3. Coste de equipos informáticos

Suponiendo la amortización de los equipos a 36 meses, podemos calcular el valor de los equipos de la siguiente manera:

Amortización/hora = 0,16 €

Total amortizable equipos = (Amortizable/hora)*Total horas trabajadas = 0,16 * 580 = **97,15 €**

En definitiva 97,15 € del coste total de los equipos son imputables al proyecto.

Coste de material fungible

COSTE DE EQUIPOS INFORMÁTICOS		
Concepto	**Cantidad**	**Precio por unidad**
Microsoft XP Profesional, SP3	2	124,48 €
Microsoft Office 2007 Profesional	2	279,92 €
Visual Paradigm for UML Standard Edition	2	250,07 €
TOTAL:		1308,94 €

Tabla 4. Coste de equipos informáticos

Suponiendo la amortización de los equipos a 36 meses, podemos calcular el valor de los equipos de la siguiente manera:

Amortización/hora = 0,22 €

Total amortizable fungible = (Amortizable/hora) * Total horas trabajadas = 0,22 * 580 = **127,6 €**

En definitiva 127,6 € del coste total del material fungible son imputables al proyecto.

Costes indirectos

Se estima que la tasa a utilizar de costes indirectos es del 15%. Como costes indirectos se consideran los gastos derivados del lugar de trabajo (posibles

alquileres), utilización de mobiliario, consumos eléctricos, etc. Por lo tanto los costes indirectos son:

Costes indirectos = Costes directos * 0,15=(9842 + 97,15 + 127,6) * 0,15=**1510,01 €**

Calculo total de costes

COSTE DE EQUIPOS INFORMÁTICOS	
Personal	9842 €
Equipos informáticos	97,15 €
Material fungible	127,6 €
Costes indirectos	1643,21 €
TOTAL:	**11709,96 €**

Tabla 5. Cálculo total de costes

Oferta de realización del proyecto

El riesgo que supone este proyecto es moderado, ya que a pesar de conocer la metodología de trabajo y la tecnología a emplear, se considera que existen factores externos que pueden causar costes adicionales. Por ello, se asume un riesgo del 15% del presupuesto.

Asimismo, se incluye en el presupuesto otro 15% adicional sobre el total que representa el beneficio estimado.

Coste totales + riesgo = 11709,96 + (11709,96 * 0,15) = 13466,45 €

Costes totales + riesgo + beneficio = 13466,45 + (13466,45 * 0,15) = 15486,42 €

Por lo tanto el precio sin IVA a pagar por los servicios prestados sería de 15486,42 €

Beneficio teórico máximo = 15486,42 - 11709,96 = 3776,46 €

Beneficio teórico mínimo = 13466,45 - 11709,96 = 1756,49 €

El precio final del proyecto con IVA (16%) es: **17964,25 €**

4. Planteamiento del problema y solución

En el presente proyecto se trata la problemática de la enseñanza de la programación orientada a objetos en Java en aquellos alumnos que reciben una primera introducción en orientación a objetos.

Se dice que la programación orientada a objetos es la más cercana a la expresión que hacemos de las cosas en la vida real en contraposición a otros tipos de programación. Pero es precisamente este pensamiento y sus conceptos básicos los que crean dicha problemática. Pues no solo consiste en pensar en una forma distinta de programar, es necesario que los alumnos adquieran conceptos básicos como son el concepto de clase, objeto, método, etc.

El origen de este proyecto viene dado por dicha problemática, la cual ha sido detectada en la asignatura de programación en Java de Ingeniería de Telecomunicación. Actualmente el método de enseñanza de programación orientada a objetos en Java consiste en una serie de clases teóricas en las que se enseña a los alumnos los principios de la programación orientada a objetos y el lenguaje de programación Java. Estas clases teóricas son apoyadas con clases prácticas con la intención de afianzar y poner en práctica dichos conceptos mediante la realización de pequeños programas (prácticas) que van aumentando el nivel a medida que aumenta el nivel de la asignatura. Dichas prácticas son realizadas con el apoyo de la herramienta JGrasp [8], un entorno de desarrollo que produce diagramas de Estructura de Control y diagramas de clase UML para Java, además de disponer de un visor de objetos y un depurador. Sin embargo, tal y como se ha comprobado a lo largo de los años, a pesar de la realización de las prácticas e incluso con el apoyo de la herramienta JGrasp [8] conceptos tan importantes como la diferenciación entre clase y objeto, la invocación de métodos y muchos otros, no quedan del todo claro para los alumnos.

Esto ha llevado a pensar en la posibilidad de que dichos conceptos pudiesen ser comprendidos de una manera más fácil mediante el uso de otras herramientas de apoyo, herramientas que proporcionen una visualización de los objetos y permitan la interacción con ellos. Es por ello que se llega finalmente así al planteamiento del presente proyecto fin de carrera con el objetivo de proporcionar un mecanismo visual que permita una representación de ciertos conceptos de la orientación objetos en Java mediante la simulación de un escenario.

4.1. Requisitos de usuario

Se han recopilado los siguientes requisitos de usuario (RU):

RU-01. Java: Los alumnos deben aprender la sintaxis del lenguaje de programación Java.

RU-02. Orientación a objetos: Los alumnos deben aprender los principales conceptos de la programación orientada a objetos.

RU-03. Práctica: Se adaptará una práctica para una herramienta que permita la visualización de los objetos, facilitando así la compresión de conceptos de la programación orientada a objetos.

RU-04. Funcionalidad-Visualización: Se separará el desarrollo funcional del juego de su visualización.

RU-05. Creación de objetos: Los alumnos deben poder realizar la creación de los objetos de manera interactiva a través de la herramienta.

RU-06. Invocación e métodos: Facilitar la compresión de la invocación a métodos mediante la interacción con los objetos.

RU-07. Acceso a las propiedades del objeto: Con el objetivo de que los alumnos puedan comprender el estado y comportamiento de los objetos, se pretende que puedan tener acceso a dicho estado de manera fácil e interactiva.

4.2. Solución

La problemática planteada junto con los requisitos de usuario definidos nos lleva a realizar un análisis más profundo de algunas herramientas de apoyo para la comprensión de los conceptos de la programación orientada a objetos. Se han estudiado aquellas herramientas desarrolladas o en proceso de desarrollo que mejor se podrían adaptar a las necesidades descritas.

Greenfoot ha sido considerada tras el estudio de las herramientas descritas en el apartado anterior, la más adecuada. A continuación se explicarán las principales razones.

La herramienta Jeliot3 ha sido descartada en primera instancia por considerarla tras su análisis una herramienta cuyo objetivo está más orientado a la visualización de algoritmos, flujos de control, arrays, creación de variables, etc. A pesar de que se puede observar la creación del objeto en la animación que se produce tras compilar el código y ejecutarlo, la diferencia entre clase y objeto visualmente no es lo suficientemente clara para el alumno.

La herramienta Alice ha sido descartada tras su análisis en primer lugar debido a que aunque se trata de una herramienta que tiene como objetivo la compresión de la programación orientada a objetos en Java, en ningún momento los alumnos pueden visualizar el código de los proyectos. Así como tampoco pueden importar las clases que ellos crearán en el entorno JGrasp, siendo éstos dos de los requisitos de mayor prioridad planteados por el usuario. Se puede destacar además de esta herramienta, la dificultad de la creación de un proyecto. Tiene una interfaz de usuario difícil de comprender incluso tras visualizar los tutoriales proporcionados por la propia herramienta, lo que hace que la motivación que se pretende buscar con el uso de la misma sea nula. Pues como es evidente, cuando es necesario "demasiado" esfuerzo para comprender el uso de una herramienta, se tiende a buscar otra con mayor facilidad o se pierde el interés.

Una vez descartadas dos de las cuatro herramientas analizadas, la decisión queda entre las herramientas BlueJ y Greenfoot. Dichas herramientas son muy parecidas, debido a que la herramienta Greenfoot, utiliza la herramienta BlueJ. Ambas tienen una interfaz de usuario fácil e intuitiva para el usuario. Pues como bien se ha explicado anteriormente no se pretende que los alumnos aprendan el uso de las herramientas sino que las herramientas les ayuden a comprender la programación orientada a objetos. La diferencia principal y decisiva que ha inclinado la balanza hacia la herramienta Greenfoot es la posesión de un mundo en el que posicionar y visualizar los objetos, es esa parte gráfica donde es posible "darle vida" a los objetos, de manera que se pueda interactuar con ellos invocando métodos y viendo el cambio resultante. De esta manera los estudiantes pueden experimentar las consecuencias de la invocación de métodos, instanciación de objetos, la comunicación entre ellos, el estado y el comportamiento. Por último destacar que la herramienta apoya la creación de juegos, para hacer divertido y atractivo su uso, cumpliendo así el requisito de motivar a los alumnos a adentrarse en la programación. Además es flexible en el sentido en que soporta distintos escenarios adaptados al grupo de usuarios a los que se quiera dirigir. Permite variar la complejidad y el nivel de dificultad de lo que se quiere enseñar y aprender.

Se muestra en la siguiente tabla (Tabla 6) resumen en la que se muestra los requisitos que cubre cada una de las herramientas analizadas:

	Jeliot3	Alice	BlueJ	Greenfoot
RU-01 (Java)	SI	NO	SI	SI
RU-02 (Orientación a objetos)	NO	NO	SI	SI
RU-03 (Práctica)	SI	NO	SI	SI
RU-04 (Funcionalidad-Visualización)	NO	NO	NO	SI
RU-05 (Creación de objetos)	NO	SI	SI	SI
RU-06 (Invocación a métodos)	NO	NO	SI	SI
RU-07 (Acceso a las propiedades del objeto)	NO	NO	SI	SI

Tabla 6. Cumplimiento Requisito - Herramienta

4.3. Trabajo a desarrollar

Una vez seleccionada la herramienta Greenfoot como la herramienta que mejor cubre los objetivos y requisitos planteados por el usuario con el objetivo de proporcionar un mecanismo visual que permita una representación de ciertos conceptos de la orientación objetos en Java mediante la simulación de un escenario. Se tiene en cuenta que el principal potencial de la citada herramienta es el desarrollo de escenarios para juegos, con la intención de motivar e incitar a continuar en el mundo de la programación.

Es por ello que se ha seleccionado el juego “Hundir la flota” para el desarrollo del escenario. Los alumnos realizarán el desarrollo del juego en primer lugar de manera independiente mediante el uso de la herramienta JGrasp, que tal y como se comentó anteriormente esta herramienta produce diagramas de Estructura de Control y diagramas de clase UML para Java, además de disponer de un visor de objetos y un depurador.

Una vez realizado el desarrollo del juego en la herramienta JGrasp se trasladará el código realizado a la herramienta Greenfoot.

El diseño del proyecto se realizará de manera que se les proporcionará unas clases ya creadas utilizando la sintaxis del api Greenfoot y realizando llamadas a los métodos que los alumnos desarrollarán, de manera que no exista la necesidad de utilizar la sintaxis del api proporcionada por Greenfoot pero que igualmente puedan visualizar los objetos. Se pretende de esta manera separar la parte funcional del juego de la parte visual.

Los alumnos tendrán que cargar el escenario que se les proporcionará y añadir el código que ellos hayan desarrollado. Seguidamente podrán interactuar con los objetos, creando objetos, invocando métodos e incluso accediendo al estado de los objetos.

4.4. Arquitectura de la aplicación

La arquitectura de la aplicación muestra la interacción entre los diferentes módulos existentes en el proceso completo para conseguir el objetivo predefinido (Figura 13). El usuario desarrollará el juego seleccionado "Hundir la flota" en la herramienta JGrasp la cual producirá archivos de extensión .java que contendrán el código fuente de las clases desarrolladas por el alumno.

Estos ficheros .java serán la entrada de Greenfoot. Serán compilados junto con el código ya proporcionado del escenario, lo que producirá como salida la visualización del juego.

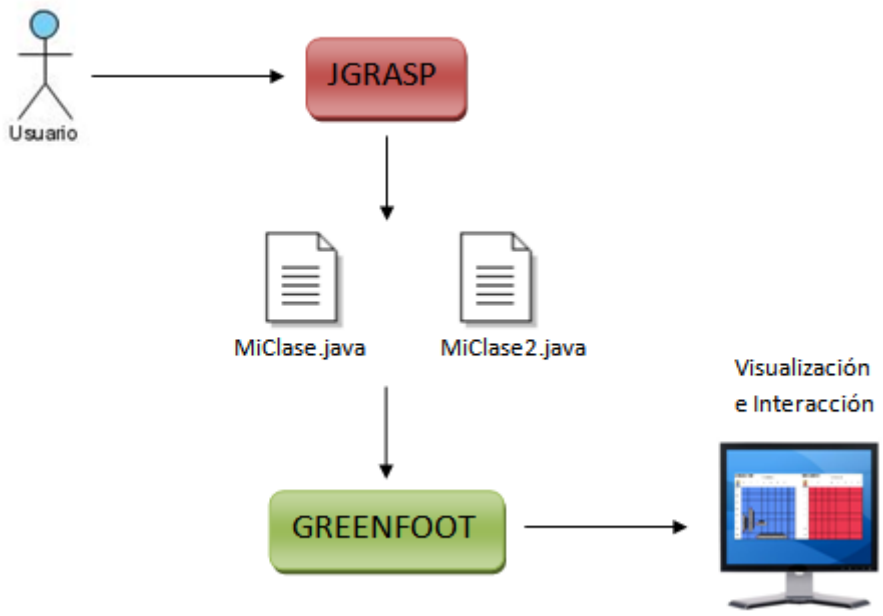


Figura 13. Arquitectura de la aplicación

5. Análisis

En este apartado comienza el proceso de desarrollo software, que abarca las etapas de Análisis, Diseño, Implementación y Pruebas.

El análisis de requisitos es el proceso de estudio de las necesidades de los usuarios para llegar a una definición de requisitos del sistema. Los requisitos son condiciones o capacidades que necesita el usuario para poder resolver un problema o conseguir un objetivo determinado. Este capítulo se centra en dichos requisitos y cómo describirlos.

Una vez definidos en el apartado anterior los requisitos de usuario, requisitos definidos de tal forma que sean comprensibles por los usuarios del sistema sin conocimiento técnico detallado, se pasa a la definición de los requisitos del sistema.

Los requisitos del sistema son versiones extendidas de los requerimientos del usuario que son utilizados por los ingenieros de software como punto de partida para el diseño del sistema. Se dividirán en:

- Requisitos Funcionales (RF)
- Requisitos No Funcionales (RNF)
- Requisitos de Usabilidad (RUS)
- Requisitos del Dominio (RD)

5.1.Juego a desarrollar

El juego seleccionado para desarrollar como escenario para la herramienta Greenfoot es el juego de “Hundir la flota”. Se ofrece a continuación una breve descripción del citado juego y las reglas básicas.

El juego estará compuesto por dos tableros uno perteneciente al jugador y otro al contrario al que llamaremos ‘máquina’.

Cada tablero estará formado por casillas las cuales podrán estar libres u ocupadas. Cada una de estas casillas tendrá sus correspondientes coordenadas. Las filas estarán representadas mediante el abecedario, comenzando por la ‘A’ y terminando por la letra correspondiente al número de filas. Las columnas sin embargo estarán representadas por número enteros comenzando por el 1.

Otro elemento importante en este juego son los buques, los cuales se posicionarán en el tablero y tendrán una longitud y una orientación determinada.

Reglas del juego

Cada jugador tendrá cuatro buques, los cuales serán posicionados de manera aleatoria. Estos cuatro buques no podrán en ningún momento cruzarse en el tablero. Dos buques no pueden ocupar una misma casilla.

En el desarrollo del juego, se alternarán los turnos. El jugador indicará la casilla que desea bombardear y el jugador contrario emite una de las tres siguientes respuestas:

- Agua: si la bomba cayó en una casilla no ocupada por un buque.
- Tocado: si la bomba cayó sobre una casilla ocupada por un buque.
- Hundido: si la comba cayó sobe un buque y todas las casillas ocupadas por éste han sido bombardeadas.

Cada una de estas situaciones será representada por una imagen distinta.

Existirá además un contador para cada tablero indicando el número de buques que quedan sin hundir.

5.2.Catálogo de requisitos

Se expondrá a continuación una serie de listados de los requisitos funcionales, no funcionales, de usabilidad y del dominio definidos como necesarios para ofrecer una solución adecuada a la problemática y al juego seleccionado para desarrollar.

5.2.1. Requisitos funcionales

Los requisitos funcionales describen lo que el sistema debe hacer. Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en la que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer.

Id	Definición
RF-01	Crear un juego, especificando el número de casillas de los tableros.

Id	Definición
RF-02	Crear las casillas que formarán el tablero.

Id	Definición
RF-03	Crear un buque que serán posicionados en el tablero.

Id	Definición
RF-04	Crear tablero compuesto por casillas.

Id	Definición
RF-05	Posicionar buques.

Id	Definición
RF-06	Jugar.

Id	Definición
RF-07	Bombardear los buques que se encuentran situados en el tablero.

5.2.2. Requisitos No Funcionales

Los requisitos no funcionales son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste. Son restricciones de los servicios o funciones ofrecidas por el sistema.

Id	Definición
RNF-01	El escenario será desarrollado para la herramienta Greenfoot versión 1.5.1.

Id	Definición
RNF-02	La herramienta podrá utilizarse en Windows, Mac, Linux y otros sistemas.

Id	Definición
RNF-03	Instalación de Java 5 o Java 6 (JDK 6) en el sistema para poder utilizar la herramienta.

5.2.3. Requisitos de Usabilidad

Los requisitos de usabilidad son requisitos que describen características y requisitos o restricciones que deben cumplir los usuarios. Son todos aquellos requisitos relacionados con el usuario.

Id	Definición
RUS-01	Conocimientos básicos de sintaxis java.

Id	Definición
RUS-02	Los usuarios no tienen conocimientos programación gráfica.

Id	Definición
RUS-03	Conocimientos mínimos de inglés para usar la herramienta Greenfoot.

Id	Definición
RUS-04	Conocimientos de orientación a objetos.

Id	Definición
RUS-05	Conocimientos mínimos de informática de usuario para poder instalar la herramienta.

5.2.4. Requisitos del Dominio

Los requisitos del dominio se derivan del dominio de la aplicación del sistema más que de las necesidades específicas de los usuarios. Son requisitos que describen restricciones que se deben aplicar solamente al escenario desarrollado con la herramienta Greenfoot.

Id	Definición
RD-01	Se mostrará un tablero para el jugador y otro para la máquina.

Id	Definición
RD-02	Cada objeto tendrá una imagen en representación del objeto, para invocar los métodos.

Id	Definición
RD-03	El tablero del jugador tendrá en todo momento los buques visibles.

Id	Definición
RD-04	El tablero de la máquina sólo mostrará los buques cuando hayan sido tocados o hundidos.

Id	Definición
RD-05	El tablero admite un máximo de 20 casillas y un mínimo de 8. Con una resolución de 1280x1024 el tamaño máximo del tablero será de 13 filas y 12 columnas para que el escenario pueda visualizarse sin necesidad de scroll.

5.3. Casos de Uso

El diagrama de Casos de Uso especifica la funcionalidad (obtenida mediante los requisitos funcionales del sistema) que el sistema ofrecerá desde la perspectiva de los usuarios y lo que el sistema realizará para satisfacer las peticiones del usuario. Cada caso de uso puede contener uno o más requisitos funcionales.

El diagrama representado a continuación en la Figura 14 muestra las que se han considerado como las principales funcionalidades que proporciona el desarrollo del escenario del juego “Hundir la flota”. El usuario, en este caso los alumnos que se introducen en la programación orientada a objetos en Java, podrá crear un juego el cual a su vez, creará dos tableros formados por casillas, creará los buques correspondientes a cada tablero y los posicionará en los mismos. Además el usuario podrá jugar mediante el bombardeo de las casillas.

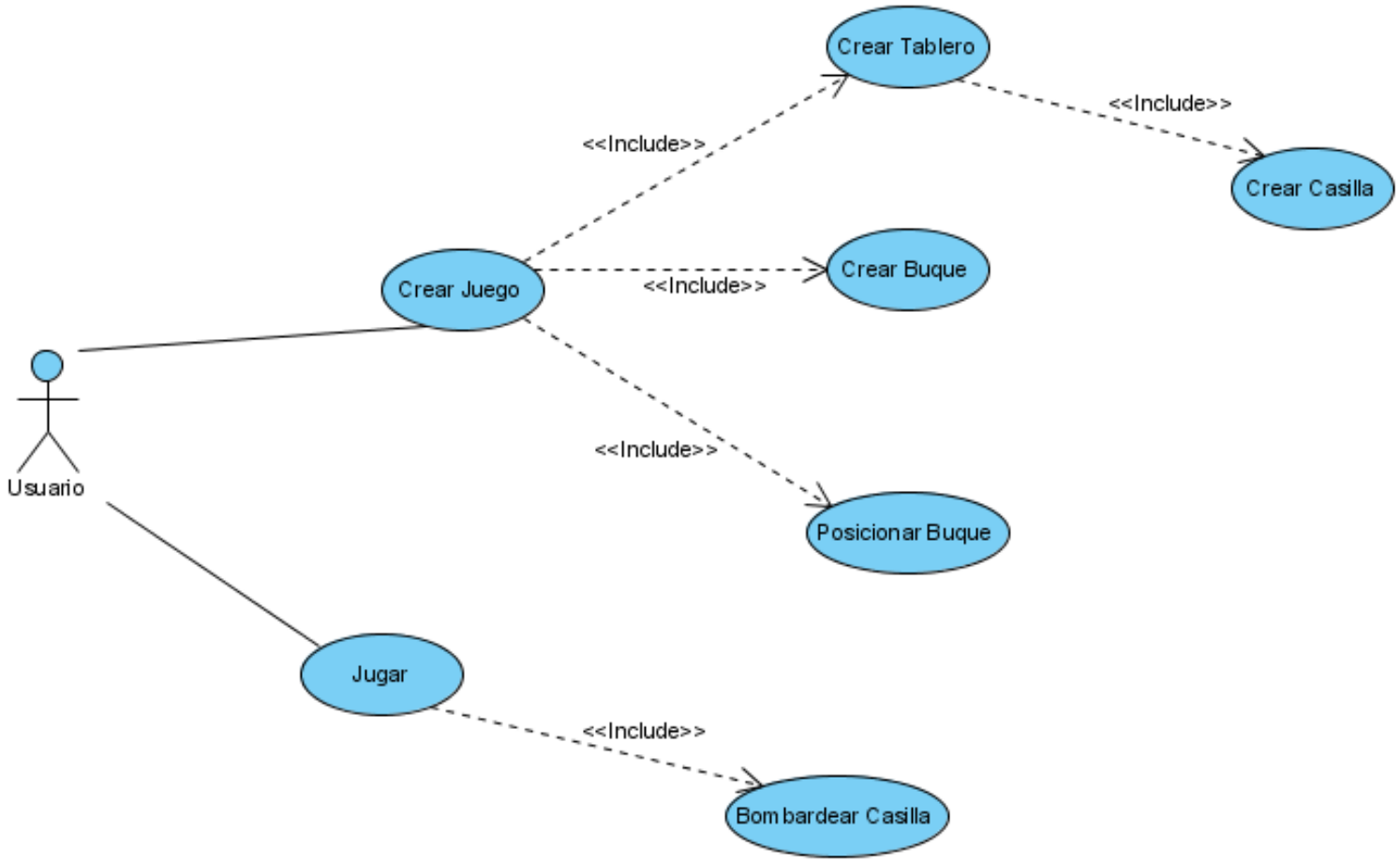


Figura 14. Diagrama de Casos de Uso

Descripción de los Casos de Uso

Id	CU-01
Caso de uso	Crear Juego
Actores	Usuario
Objetivo	La creación de un nuevo juego, bien sea un juego por defecto con el tamaño de los tableros 8x8 o un juego en el que el usuario decida el número de filas y casillas de los tableros
Precondiciones	La clase Juego previamente creada e implementada.
Poscondiciones	Se proporcionará un nuevo juego listo para jugar.
Escenario básico	En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Juego el cual creará dos tableros, uno para el jugador y otro para la máquina, creará los buques correspondientes y posicionará los cuatro buques en cada uno de los tableros, de manera aleatoria.
Requisitos	RF-01

Id	CU-02
Caso de uso	Crear Tablero
Actores	Usuario
Objetivo	Crear los tableros que formarán el juego. Un tablero para el jugador y otro para la máquina.
Precondiciones	La clase Tablero previamente creada e implementada.
Poscondiciones	El tablero creado formará parte del juego y serán

	posicionados en él los buques.
Escenario básico	Se creará un array bidimensional de casillas.
Requisitos	RF-04

Id	CU-03
Caso de uso	Crear Casilla
Actores	Usuario
Objetivo	Crear las casillas que formarán el/los tablero/s
Precondiciones	La clase Casilla debe estar previamente creada e implementada.
Poscondiciones	La casilla creada formará parte de un tablero.
Escenario básico	Se realizará la invocación del constructor de la clase Casilla el cual creará la casilla.
Requisitos	RF-02

Id	CU-04
Caso de uso	Crear Buque
Actores	Usuario
Objetivo	Crear buques que serán posicionados en el tablero.
Precondiciones	La clase Buque previamente creada e implementada.
Poscondiciones	El buque creado será posteriormente posicionado en el tablero.
Escenario básico	Se realizará la invocación del constructor de la

	clase Buque que recibirá como parámetros el nombre, la longitud y la orientación de dicho buque y lo creará en el mundo.
Requisitos	RF-03

Id	CU-05
Caso de uso	Posicionar buque
Actores	Usuario
Objetivo	Posicionar los buques en el tablero para poder iniciar un juego
Precondiciones	Tablero creado Buque creado
Poscondiciones	Buques posicionados sobre el tablero
Escenario básico	En primer lugar se comprobará si las casillas a ocupar por el buque se encuentran dentro de los límites del tablero, y si la orientación del buque es válida. En caso de ser verdaderas las dos condiciones anteriores, se comprobará en segundo lugar si se puede posicionar en dichas casillas, para ello se procede a mirar si las casillas no contienen otro buque y si el citado buque no se cruza con otro buque posicionado anteriormente. Por último, siendo afirmativas también las condiciones anteriores, se alojará el buque en las casillas correspondientes.
Requisitos	RF-05

Id	CU-06
Caso de uso	Jugar
Actores	Usuario
Objetivo	Jugar una partida a juego 'Hundir la flota'
Precondiciones	Juego previamente creado, y buques posicionados
Poscondiciones	Inicio o fin de la partida
Escenario básico	En este caso existen dos posibilidades: 1. Jugar automáticamente: el usuario invocará al método jugar() el cual bombardeará aleatoriamente, alternando turno, primero en un tablero (el del jugador, por ejemplo) y luego en el otro, mostrando visualmente si se ha alcanzado un buque o no. 2. Jugar interactuando: el usuario pulsará el botón run y hará click, en el tablero de la máquina, en la casilla que quiera bombardear y automáticamente se bombardeará aleatoriamente el tablero de la máquina. En este caso también se mostrará visualmente si se ha alcanzado un buque o no.
Requisitos	RF-06

Id	CU-07
Caso de uso	Bombardear Casilla
Actores	Usuario
Objetivo	Bombardear los buques del jugador contrario.
Precondiciones	Tableros creados.

	Buques creados y posicionados sobre los tableros. Juego creado.
Poscondiciones	Buque bombardeado, posibilidad de fin del juego.
Escenario básico	En primer lugar se comprobará si la casilla se encuentra dentro de los límites del tablero. En caso de que se encuentre en los límites del tablero, se comprobará que dicha casilla no ha sido bombardeada con anterioridad. Si dicha casilla no ha sido bombardeada anteriormente, se bombardeará la casilla.
Requisitos	RF-07

6. Diseño

Una vez definidas las necesidades de la aplicación relativas a la solución a ofrecer, en este apartado del documento se analizará la estructura con la que modelar dicha solución.

El escenario a desarrollar es el famoso y clásico juego 'Hundir la flota' en el que un jugador se enfrenta al ordenador (máquina) en una guerra por hundir sus buques. Dichos buques serán colocados en los tableros de tamaño NxM, de forma automática y aleatoria. El número de buques por tablero (jugador y máquina) será cuatro, una Fraga de longitud 2, un Destructor de longitud 3, un Crucero de longitud 4 y un PortaAviones de longitud 5. El ganador será aquel que consiga hundir todos los buques del contrario.

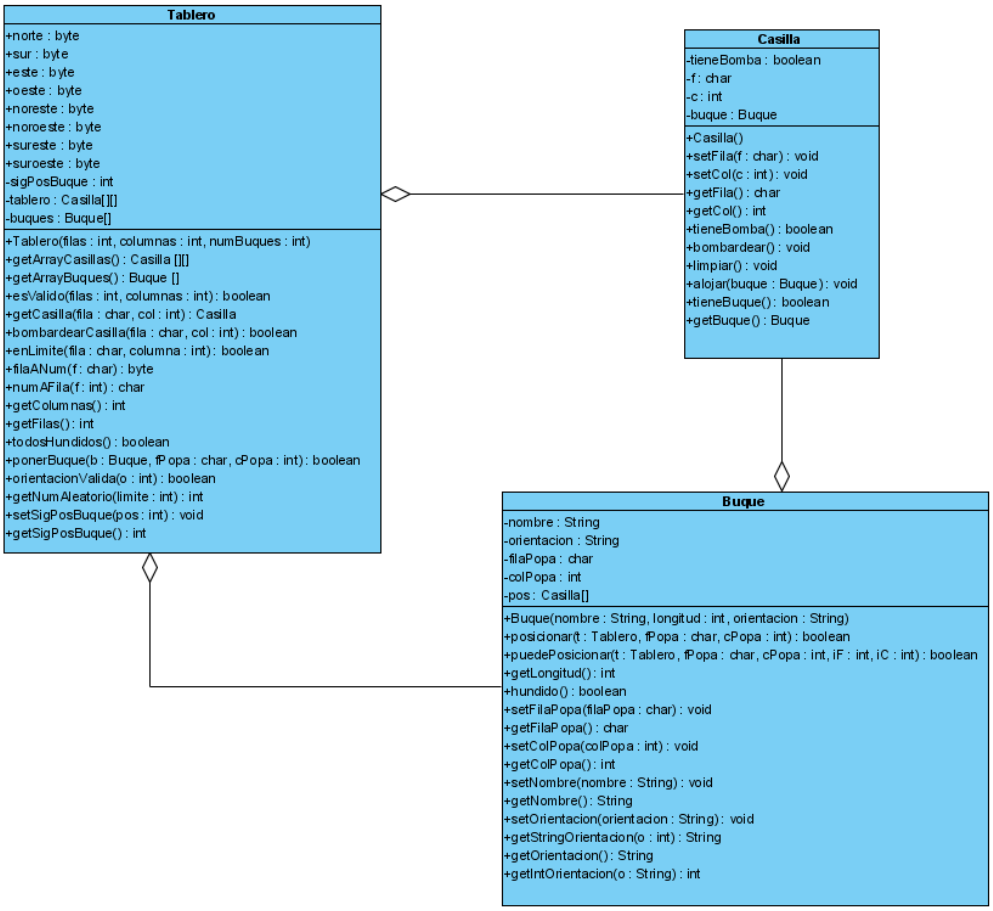


Figura 15. Diagrama de clases

Tras analizar las reglas del juego y las funcionalidades requeridas, la Figura 15 muestra el diagrama de clases con las clases que se han considerado necesarias para el desarrollo del juego y sus relaciones. Como puede observarse, se dispone de las siguientes clases:

- **Casilla**: que representa un área del mar donde se podrán alojar buques y bombas. Almacenará información indicando si dicha casilla esta bombardeada o no, así como el buque que está ocupando esa casilla y la fila y la columna correspondiente a la casilla. Los métodos de esta clase son:
  - *setFila():* Asigna la fila a la casilla.
  - *getFila():* Devuelve la fila de la casilla.
  - *setCol():* Asigna la columna de la casilla.
  - *getCol():* Devuelve la fila de la casilla.
  - *tieneBomba():* Devuelve TRUE si la casilla ha sido bombardeada y FALSE en caso contrario.
  - *bombardear():* Arroja una bomba sobre la casilla.
  - *limpiar():* Limpia las bombas de una casilla y desaloja el buque que pudiera estar ocupándola.
  - *alojar():* Aloja el buque indicado sobre la casilla, de forma que la casilla conoce que buque la ocupa.
  - *tieneBuque():* Devuelve TRUE si sobre la casilla hay un buque alojado y FALSE en caso contrario.
  - *getBuque():* Devuelve el buque alojado en la casilla.
- **Tablero**: representa el tablero como un conjunto de casillas y un conjunto de buques. Los métodos de esta clase son:
  - *bombardearCasilla(char fila, int col):* Arroja una bomba en la casilla(fila,col). Devuelve TRUE si la casilla ha sido bombardeada y FALSE en caso contrario.
  - *enLimite(char fila, int columna):* Comprueba que la fila y la columna indicadas se encuentran dentro de los límites del

tablero, devolviendo TRUE en ese caso y FALSE en caso contrario.

- *esValido(int filas, int columnas):* Comprueba si el número de filas y columnas del tablero se encuentran dentro de los límites mínimo 8 y máximo 20. Devuelve TRUE de ser así y FALSE en caso contrario.
- *filaANum(char f):* Toma un carácter y devuelve el número correspondiente al índice representado por ese carácter, de forma que la 'A' corresponde con 0, la 'B' con 1, etc.
- *numAFila(int f):* Toma un número y devuelve el carácter correspondiente, de forma que el 0 se corresponde con la 'A', el 1 con la 'B', etc.
- *getArraysBuques():* Devuelve el array de buques correspondiente al tablero.
- *getArrayCasillas():* Devuelve el array de casillas correspondiente al tablero.
- *getCasilla(char fila, int col):* Devuelve el objeto casilla situado en la fila y columnas indicadas. La fila se indica con un carácter tal que la primera fila será la 'A'.
- *getColumnas():* Devuelve el número de columnas que tiene el tablero.
- *getFilas():* Devuelve el número de filas que tiene el tablero.
- *getNumAleatorio(int limite):* Devuelve un número aleatorio dentro del límite proporcionado.
- *getSigPosBuque():* Devuelve el número de buques posicionados en el tablero.
- *orientacionValida(int o):* Devuelve TRUE si la orientación es válida y FALSE en caso contrario.
- *ponerBuque(Buque b, char fPopa, int cPopa):* Trata de posicionar el buque b en el tablero, de tal forma que la popa se ubique en la fila fPopa y la columna cPopa.

- *setSigPosBuqe():* Establece el número de buques añadidos hasta el momento.
- *todosHundidos():* Devuelve TRUE si todos los buques del tablero están hundidos y FALSE en caso contrario.

- **Buque**: el tablero de cada jugador contendrá varios buques de guerra. Cada uno de ellos tiene una longitud que viene dada por el número de casillas contiguas que éste ocupa en una determinada dirección (N, S, E, O, NO, SO, NE, SE, como se indica en la Figura 16).

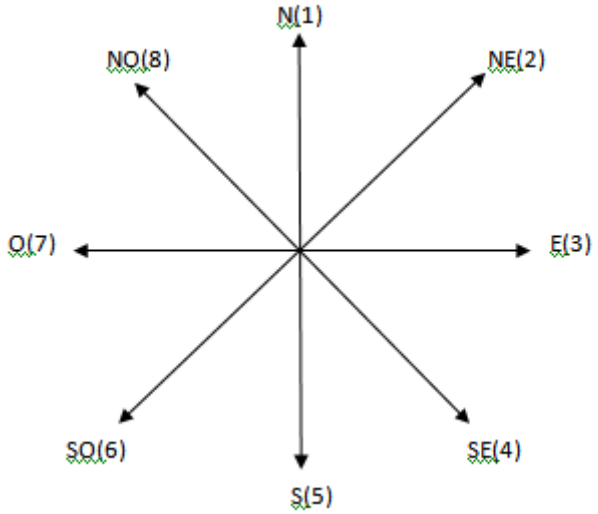


Figura 16. Orientación válida de los buques

Los métodos de esta clase son:

- *getColPopa():* Devuelve la columna de la popa del buque.
- *getFilaPopa():* Devuelve la fila de la popa del buque.
- *getIntOrientacion(String o):* Toma un String y devuelve su orientación correspondiente.
- *getLongitud():* Devuelve la longitud del buque.
- *getNombre():* Devuelve el nombre del buque.

- *getStringOrientacion(int o):* Toma un entero y devuelve su orientación correspondiente.
- *hundido():* Devuelve TRUE si el buque está hundido, y FALSE en caso contrario.
- *posicionar(Tablero t, char fPopa, int cPopa):* Posiciona el buque en el tablero.
- *puedePosicionar(Tablero t, char fPopa, int cPopa, int iF, int iC):* determina si el buque puede situarse en el tablero t con la popa en la casilla (fPopa, cPopa).
- *setColPopa(int colPopa):* Establece la columna de la popa del buque.
- *setFilaPopa(char filaPopa):* Establece la fila de la popa del buque.
- *setNombre(String nombre):* Establece el nombre del buque.
- *setOrientacion(String orientación):* Establece la orientación del buque.
- *getOrientacion():* Devuelve la orientación del buque.

Tal y como se observa en las relaciones entre clases en el diagrama de clases presentado en la Figura 15, un tablero estará formado por casillas y buques y un buque estará formado a su vez por un conjunto de casillas.

Teniendo en cuenta la necesidad del uso del api Greenfoot, para la visualización de los objetos, la idea inicial en el diseño del juego era que los alumnos una vez creado el código en su herramienta normal de trabajo, intercalasen dicho código con el desarrollado con la sintaxis de Greenfoot para la visualización de los objetos. Sin embargo, una vez iniciado el proceso de implementación, se llegó a la conclusión de que esto podría reducir la intención de motivar a los alumnos para profundizar en el mundo de la programación orientada a objetos en Java, surgiendo así un nuevo requisito de usuario con la intención de separar la parte funcional del juego de la parte visual. Por lo que se tomó la decisión de rediseñar el juego. Este diseño será comentado en el capítulo de Implementación debido a que se trata de una cuestión más de implementación que de diseño.

7. Implementación

En este apartado se explicará la decisión de modificar el diseño del diagrama de clases del juego para adoptarlo a la herramienta Greenfoot con la necesidad de separar la parte funcional del juego de la parte gráfica.

Esta necesidad viene dada por las restricciones del uso del API Greenfoot. Las clases cuyos objetos se quieran visualizar deben heredar de la clase Greenfoot-World o Greenfoot-Actor debido a que proporcionan métodos para establecer las imágenes de los objetos, cambiar el tamaño del mundo, dibujar sobre el fondo, etc.

Para evitar que los alumnos utilicen del API de Greenfoot, debido a que esto podría disminuir el inicial interés en el uso de la herramienta, además de las clases descritas anteriormente, se han creado clases paralelas a las que llamamos Casilla_Greenfoot, Tablero_Greenfoot y Buque_Greenfoot que usan las clases desarrolladas por los alumnos y la sintaxis de Greenfoot. Los métodos públicos de estas clases utilizan los implementados por el alumno en las clases Casilla, Tablero y Buque.

7.1. Diagrama de clases

El diagrama de clases resultante se muestra en la Figura 17 a continuación. Como puede observarse las clases creadas por los tableros mantienen las mismas relaciones, y las clases paralelas Casilla_Greenfoot, Tablero_Greenfoot y Buque_Greenfoot tienen entre ellas la misma relación que las clases desarrolladas por los alumnos con las cuales tienen una relación de uso. Por ejemplo: la clase Casilla_Greenfoot, usa la clase Casilla debido a que invoca a sus métodos y crea dentro de ella un objeto Casilla.

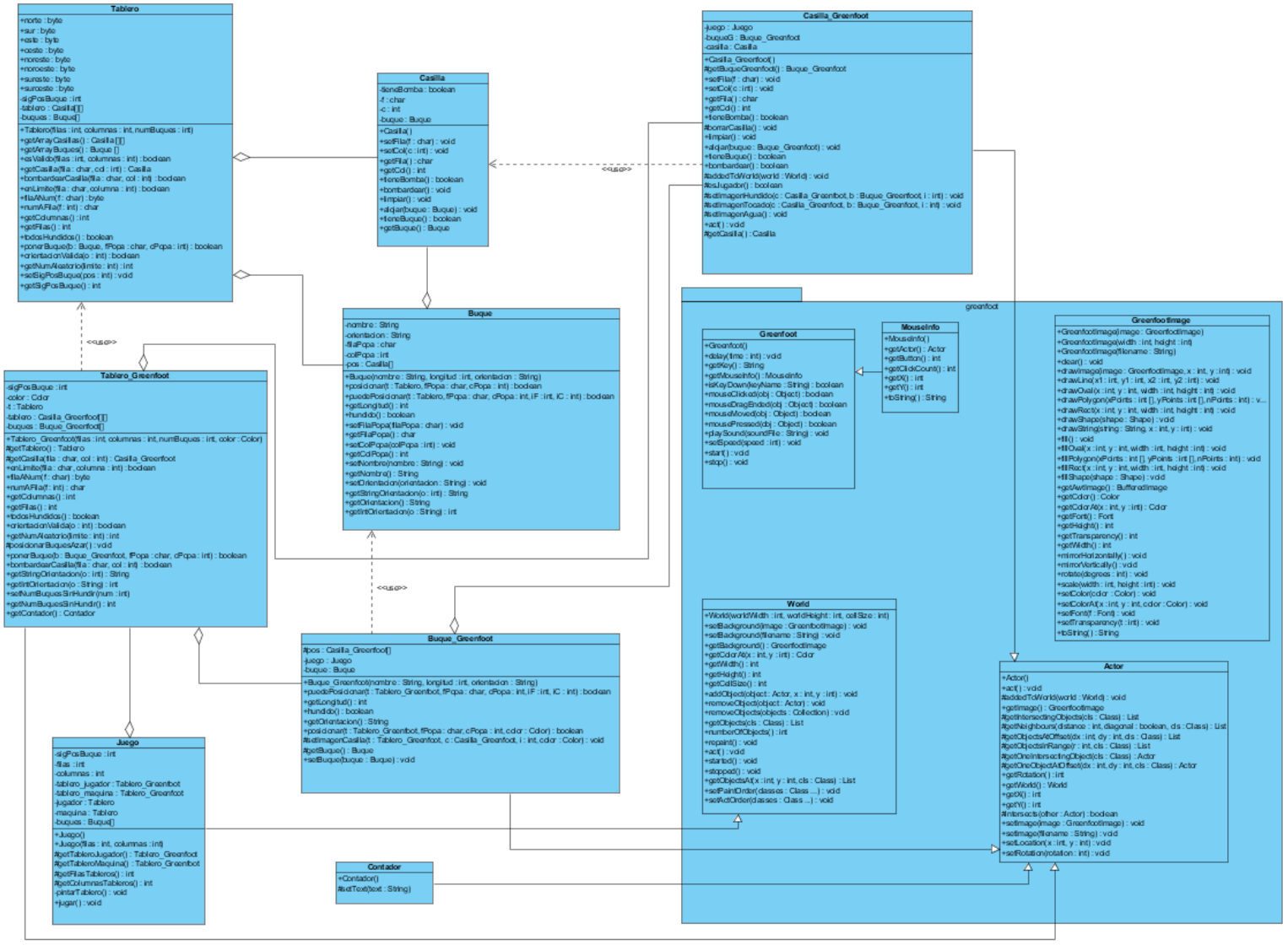


Figura 17. Diagrama de clases con Greenfoot

Se ofrece a continuación una descripción de cada una de las clases, excepto las correspondientes al paquete de Greenfoot (Greenfoot, GreenfootImage, Actor, MouseInfo y World) y las desarrolladas por los alumnos comentadas en el apartado de diseño:

- **Casilla_Greenfoot**: esta clase representa la misma información que la clase Casilla explicada en el apartado anterior, sin embargo permite la representación de la casilla en el mundo de los objetos mediante la asignación de imágenes con métodos como:
    - setImagenAgua(): proporcionando la imagen correspondiente de la casilla en caso de que ésta haya sido bombardeada y no tenga ningún buque alojado.
    - setImagenHundido(): que establece la imagen de la casilla en caso de que dicha casilla contenga un buque y éste haya sido hundido (bombardeado en su totalidad).
    - setImagenTocado(): asigna la imagen de la casilla en caso de que la casilla tenga alojado un buque y éste haya sido bombardeado.
- **Tablero_Greenfoot**: representa la misma información que la clase Tablero, sin embargo añade métodos y sintaxis que permiten su visualización y representación en el mundo de los objetos.
- **Buque_Greenfoot**: representa la misma información que la clase Buque pero añade métodos (como 'setImagenCasilla') y sintaxis que permiten su representación.
- **Juego**: esta clase es la encargada de representar el juego completo, formado por dos tableros uno correspondiente al jugador y otro a la máquina. Es el encargado de pintar el juego.
- **Contador**: esta clase representa los contadores de cada tablero. Llevarán una cuenta atrás del número de buques que quedan en cada tablero. Esto podría realizarse en cualquier entorno de una manera más sencilla, sin necesidad de crear una clase específica para ello, sin embargo es la sintaxis de Greenfoot, la visualización, el hecho de tener que cambiar el valor del contador lo que hace que sea necesario crear una clase específica para esto.

7.2. Diagramas de secuencia

Para facilitar la comprensión de la decisión del diseño de clases mostrado anteriormente, se mostrará a continuación detalles de implementación de los casos de uso con mayor complejidad, incluyendo los objetos y clases usados, y mensajes intercambiados entre los objetos. Se determinará así los objetos que son necesarios para la implementación.

Los casos de uso que serán representados mediante diagrama de secuencia son:

- **Crear Juego** (Figura 18)
- **Posicionar Buque** (Figura 19)
- **Jugar** (Figura 20)
- **Bombardear Casilla** (Figura 21)

Crear Juego

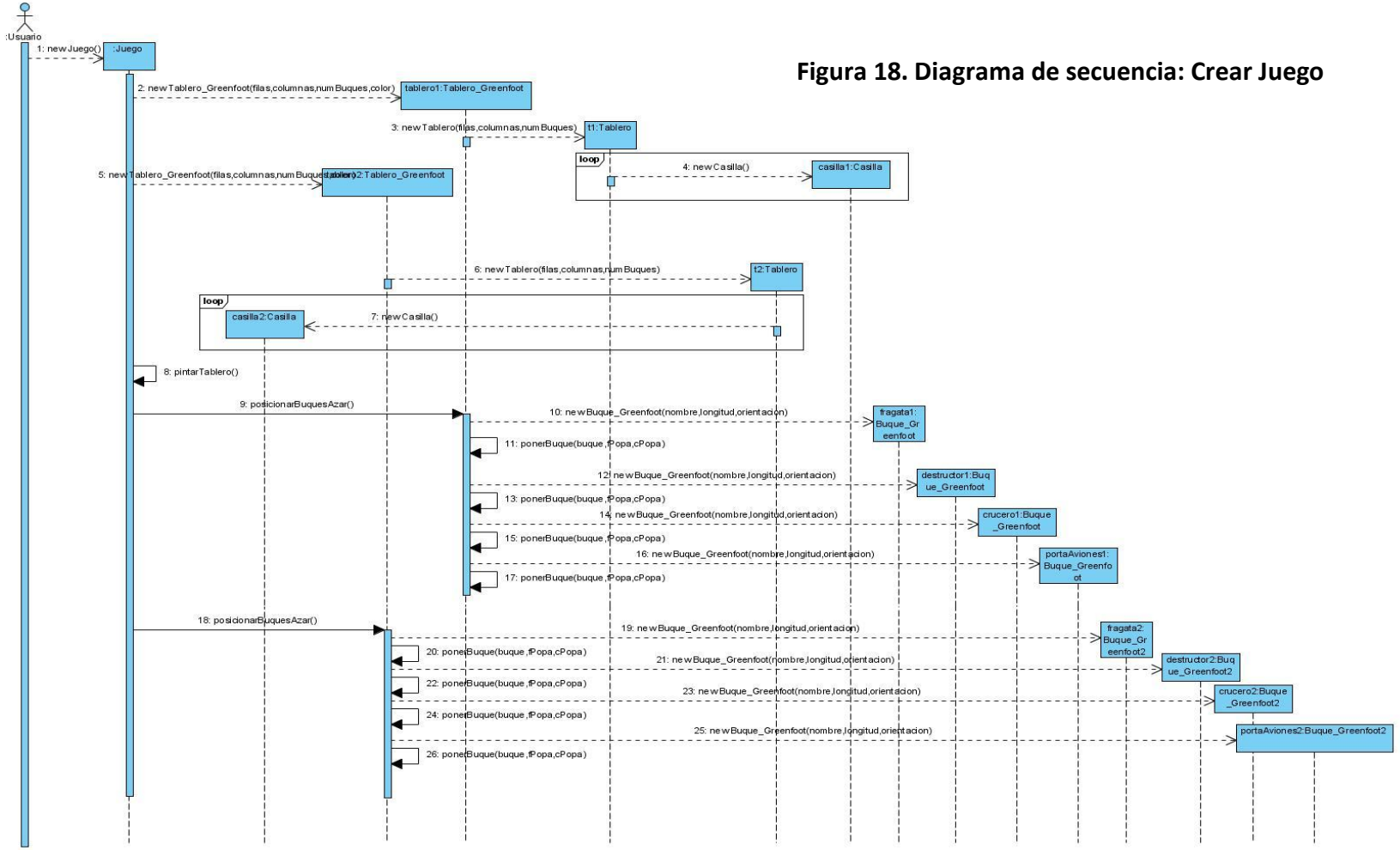
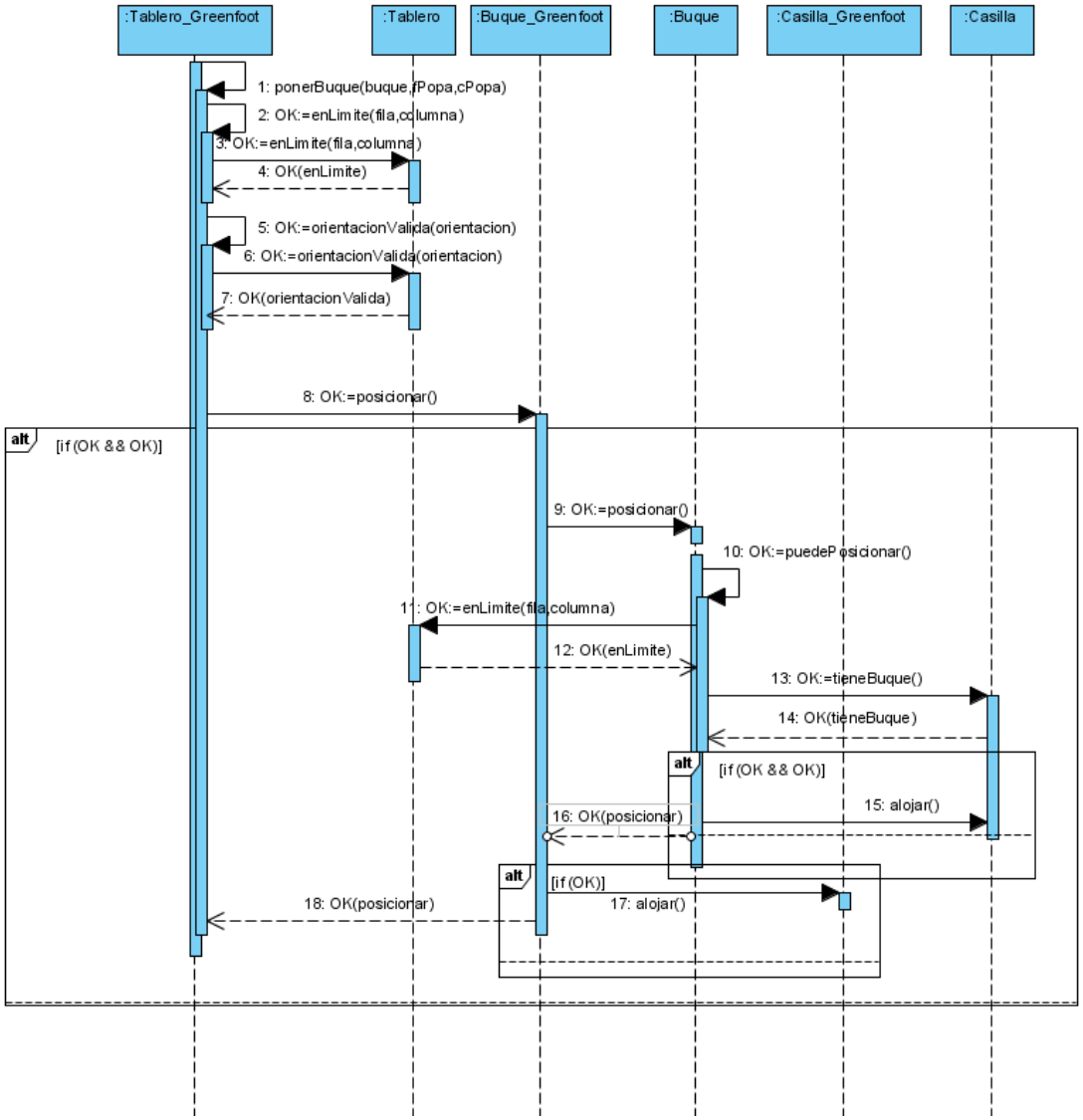


Figura 18. Diagrama de secuencia: Crear Juego

La secuencia empieza cuando el usuario invoca el constructor de la clase juego. Seguidamente se crearán dos tableros uno para el jugador y otro para la máquina, que estarán formados por un array de casillas. Después se pintarán los tableros en el mundo y se pasará a posicionar los buques al azar en cada uno de los tableros. Para ello se crearán cuatro buques y se invocará al método “ponerBuque” cuya interacción se describe en el diagrama de secuencia ‘Posicionar Buque’ de la Figura 19.



Posicionar Buque

La secuencia empieza cuando el método "ponerBuque" encargado de posicionar el buque es invocado, se realizan una serie de comprobaciones para posicionar el buque. En primer lugar se comprueba que el buque que se quiere posicionar, se pretende posicionar dentro de los límites del tablero. En segundo lugar se comprueba que la orientación que llevará el buque es válida. En caso de ser así se comprobará si se puede posicionar mirando que el buque no se cruce con ningún otro buque posicionado anteriormente, para ello se comprueba a medida que se va avanzando en el tablero si la casilla en la que se pretende posicionar está en los límites y si contiene otro buque. En caso de estar libre y que el buque no se cruce con otro, se alojará el buque.

Figura 19. Diagrama de secuencia: Posicionar Buque

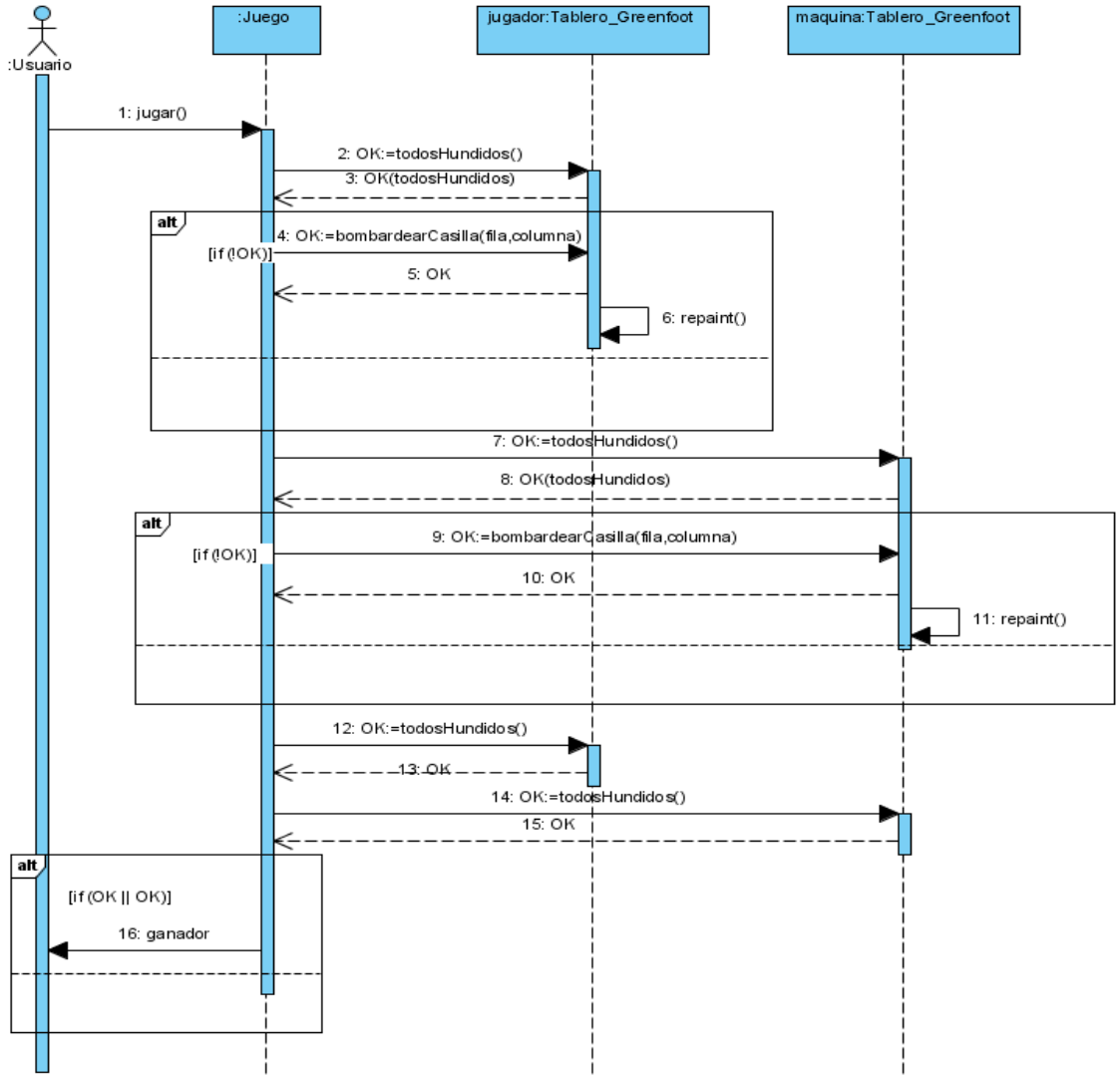


Figura 20. Diagrama de secuencia: Jugar

Jugar

La secuencia empieza cuando el jugador invoca el método "jugar" perteneciente a la clase Juego. Se comprobará en una primera instancia si los buques del jugador han sido ya hundidos. De no ser así se dispondrá a bombardear una casilla (explicado en el diagrama de la Figura 21) del citado tablero cuya fila y columna serán adquiridas de manera aleatoria. En segundo lugar se producirá la misma secuencia para el tablero de la máquina. Y estas secuencias se repetirán hasta el momento en que uno de los dos tableros tenga todos los buques hundidos, mostrando así el ganador del juego y finalizándolo.

Bombardear Casilla

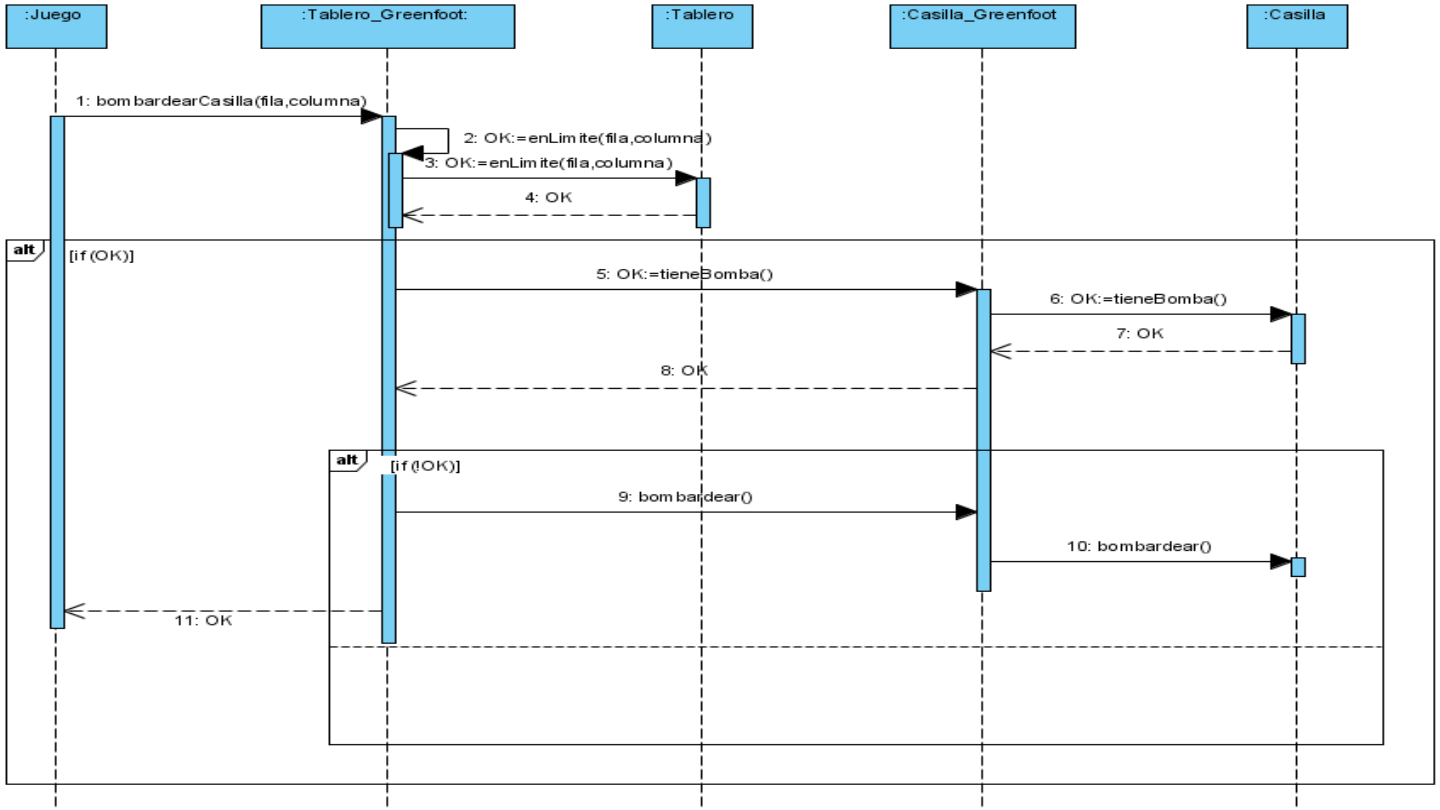


Figura 21.Diagrama de secuencia: Bombardear Casilla

El método "bombardearCasilla" será invocado por el juego y seguirá la siguiente secuencia. Comprobará que la casilla se encuentra en los límites del tablero, seguidamente si dicha casilla tiene una bomba ya alojada. En caso de no tener una bomba alojada, se bombardeará la casilla.

Todos los requisitos y modelos definidos a lo largo del documento tienen como único objetivo asegurar una sencilla, correcta y valida implementación de la solución.

Resulta obvio considerar que en el presente documento no se incluirá código fuente. La documentación electrónica que se adjunte al documento escrito contendrá el conjunto del código fuente por si fuese necesaria su consulta.

7.3. Herramientas

La herramienta utilizada para el desarrollo del proyecto es la herramienta Greenfoot, la cual se escogió tras el análisis realizado sobre otras herramientas existentes en el mercado.

Esta herramienta, tal y como se explicó anteriormente en el documento, nos ofrece una interfaz visual dividida en tres partes principales.

La parte central, en la que se realiza la visualización de los objetos y su interacción, la parte derecha en la que se muestra un navegador de clases que aporta una vista de las clases que participan en la simulación del escenario. Estas clases pueden ser editadas, compiladas e instanciadas. Acciones a las que accederemos mediante el menú pop-up de las clases. Por último en la parte baja de la herramienta se proporcionan una serie de botones para el control de la ejecución. Con ellos se puede ejecutar, parar o simular un solo paso de ejecución. Además hay un deslizador para controlar la velocidad de ejecución.

Además de la interfaz visual, Greenfoot nos ofrece un entorno de desarrollo integrado consistente en un editor, un compilador y un depurador de código. Proporcionando todas las herramientas necesarias para el desarrollo, examen y ejecución de una aplicación completa.

7.4. Organización del Código

En este apartado destacaremos la organización del código. Greenfoot nos ofrece su propia y obligada organización del código mostrado en el navegador de clases. Existe una distinción entre las clases que heredan de la clase World, las clases que heredan de la clase Actor y el resto de las clases. En la siguiente imagen (Figura 22) se muestra tal organización:

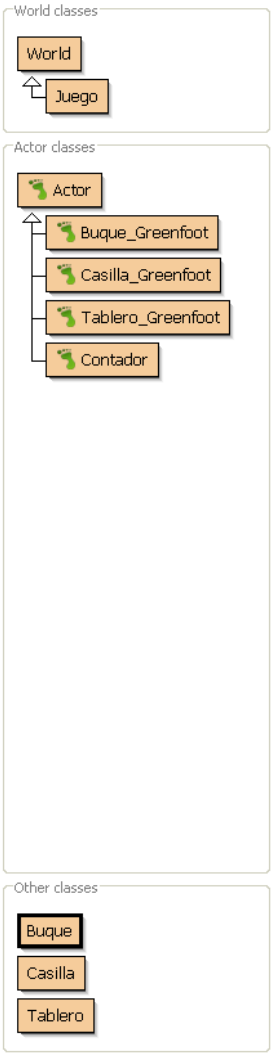


Figura 22. Organización del código

7.5. El producto del desarrollo

Para finalizar con el capítulo, se mostrarán a continuación una serie de imágenes sobre las pantallas más representativas del sistema. Con ello se puede obtener una idea más precisa acerca del proyecto desarrollado. Las imágenes incluyen anotaciones sobre ciertos elementos o funcionalidades concretas que suponen un aporte significativo en la aplicación.

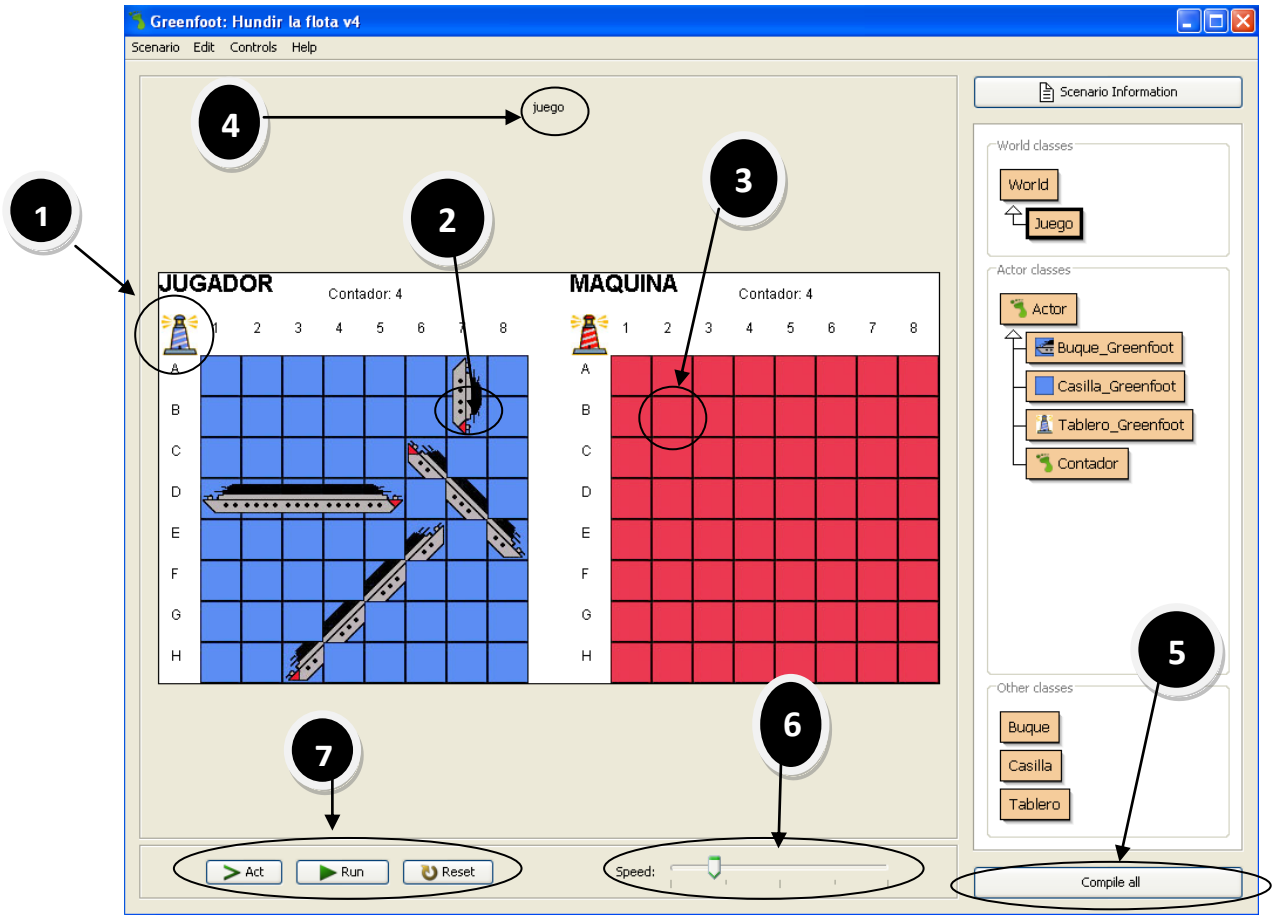


Figura 23. Juego por defecto

En esta la imagen anterior (Figura 23) se muestra la pantalla de un juego creado por defecto en el que los tableros tienen una dimensión de 8x8 casillas. Se han señalado algunos de los elementos considerados los más importantes:

1. Imagen del faro, para invocar los métodos de la clase Tablero_Greenfoot
2. Imagen del buque con un triángulo rojo para invocar los métodos de la clase Buque_Greenfoot.
3. Imagen que representa la casilla. En color rojo representa una casilla perteneciente al tablero de la máquina y en azul al tablero del jugador. Se usarán para invocar los métodos de la clase Casilla_Greenfoot
4. Esta etiqueta representa la instancia de la clase Juego desde la que se invocan sus métodos.

5. Botón para compilar. Se compila así todas las clases que aparecen en el menú de clases.

6. Regulador de velocidad de ejecución. Podemos aumentar o reducir con él la velocidad de ejecución.

7. Botones para controlar la ejecución. Encontramos entre ellos el botón Act, que en este caso no tendrá ningún tipo de utilidad, el botón Run, que se utilizará para iniciar la ejecución y poder jugar interaccionando y por último el botón Reset que nos permitirá empezar un juego nuevo.

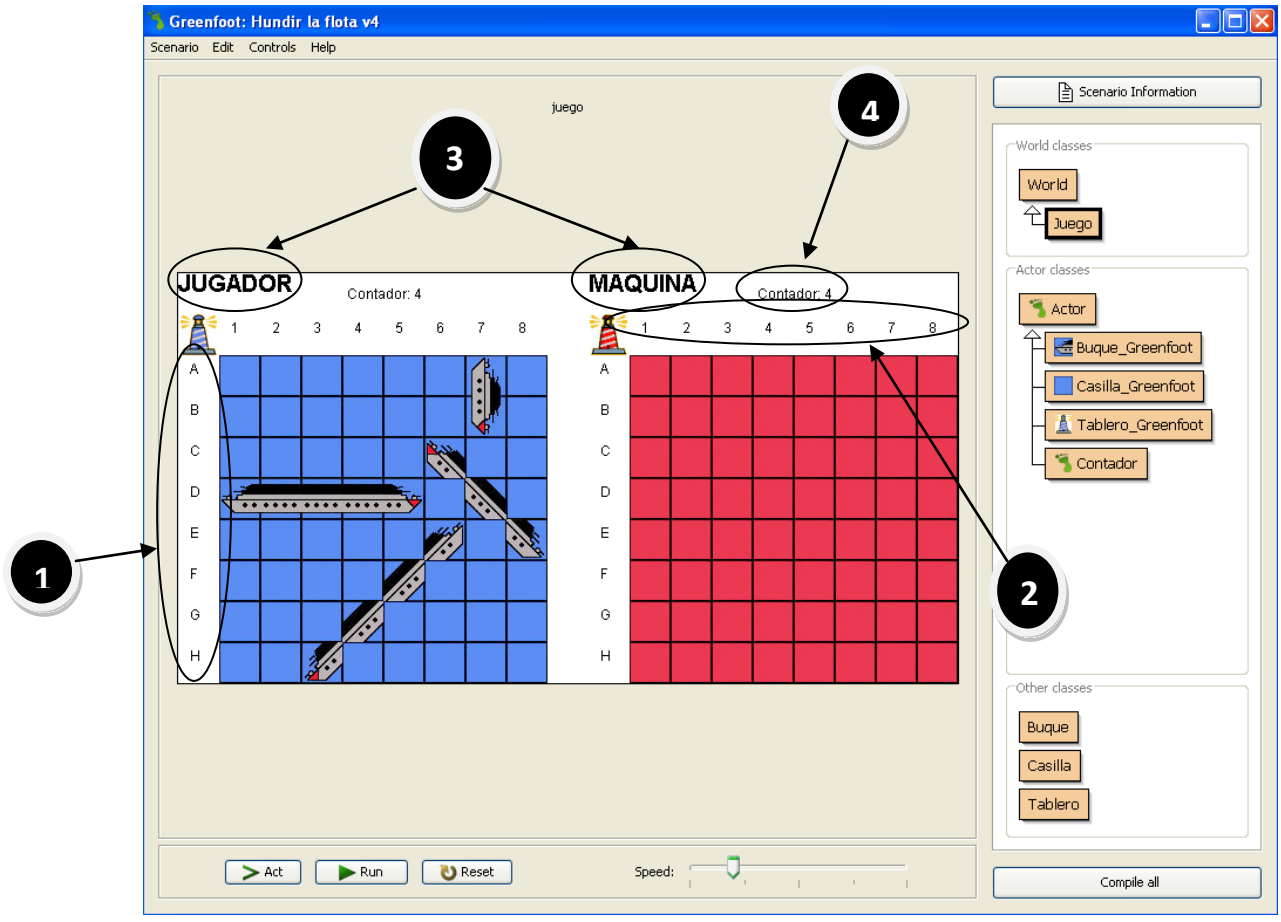


Figura 24. Juego definido por el usuario

En la imagen anterior Figura 24, se muestra un juego creado por el usuario, en el que el usuario introduce las dimensiones del juego que desee, en este caso los

tableros tienen un tamaño de 12x12. Se señalan algunos elementos importantes para el desarrollo del juego:

1. Coordenadas de las filas. A cada fila le corresponde una letra desde la 'A' hasta la letra correspondiente al número de fila.
2. Coordenadas de las columnas. Su numeración comienza en 1.
3. Jugadores. Indica a qué jugador pertenece cada tablero.
4. Contador de buques que quedan en el tablero sin hundir. Existe uno para cada tablero, e irá actualizándose a medida que se hunda un buque.

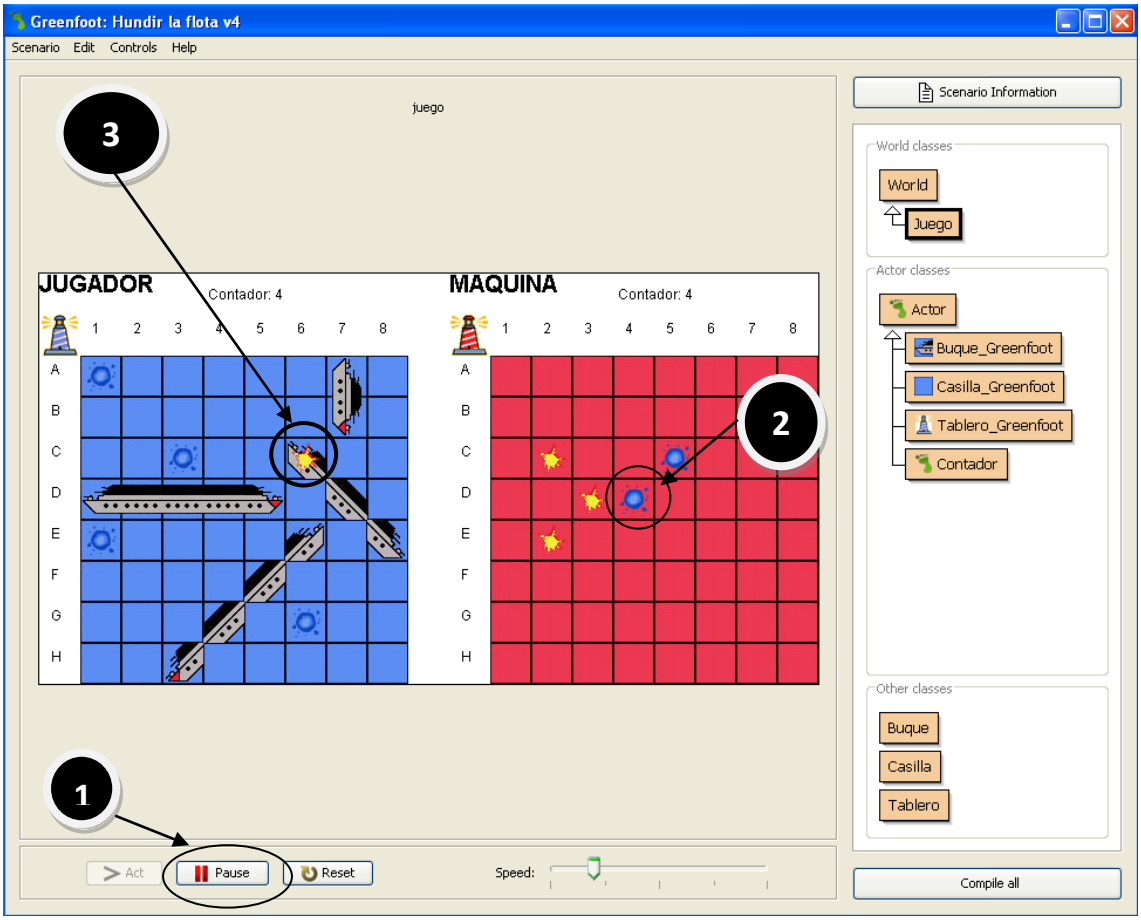


Figura 25. Ejecución del juego

En esta imagen (Figura 25) se muestra una pantalla con la ejecución del juego en la que el usuario interactúa. El usuario clickea en la casilla que quiere bombardear y el tablero del jugador es bombardeado aleatoriamente. Se muestran algunos elementos importantes:

1. Botón de ejecución activado para poder jugar (anteriormente debe aparecer nombrado como Run).

2. Imagen que representa una casilla bombardeada que no tiene buque

3. Imagen que representa una casilla bombardeada que contiene un buque

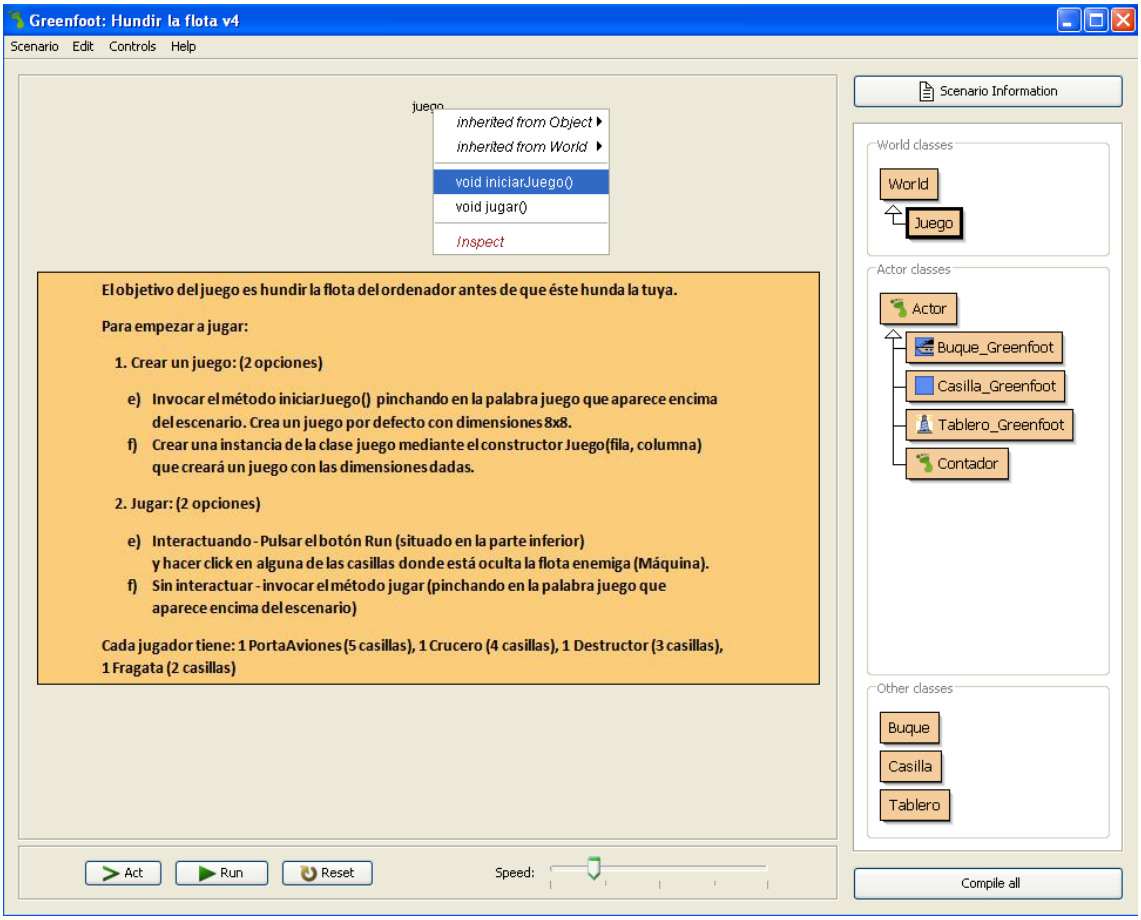


Figura 26. Creación de un juego por defecto con tablero 8x8

En esta pantalla (Figura 26) se muestra como el usuario crea un nuevo juego por defecto en el que los tableros tendrán cada uno un tamaño de 8x8 casillas. Inicialmente aparecerán las instrucciones del juego.

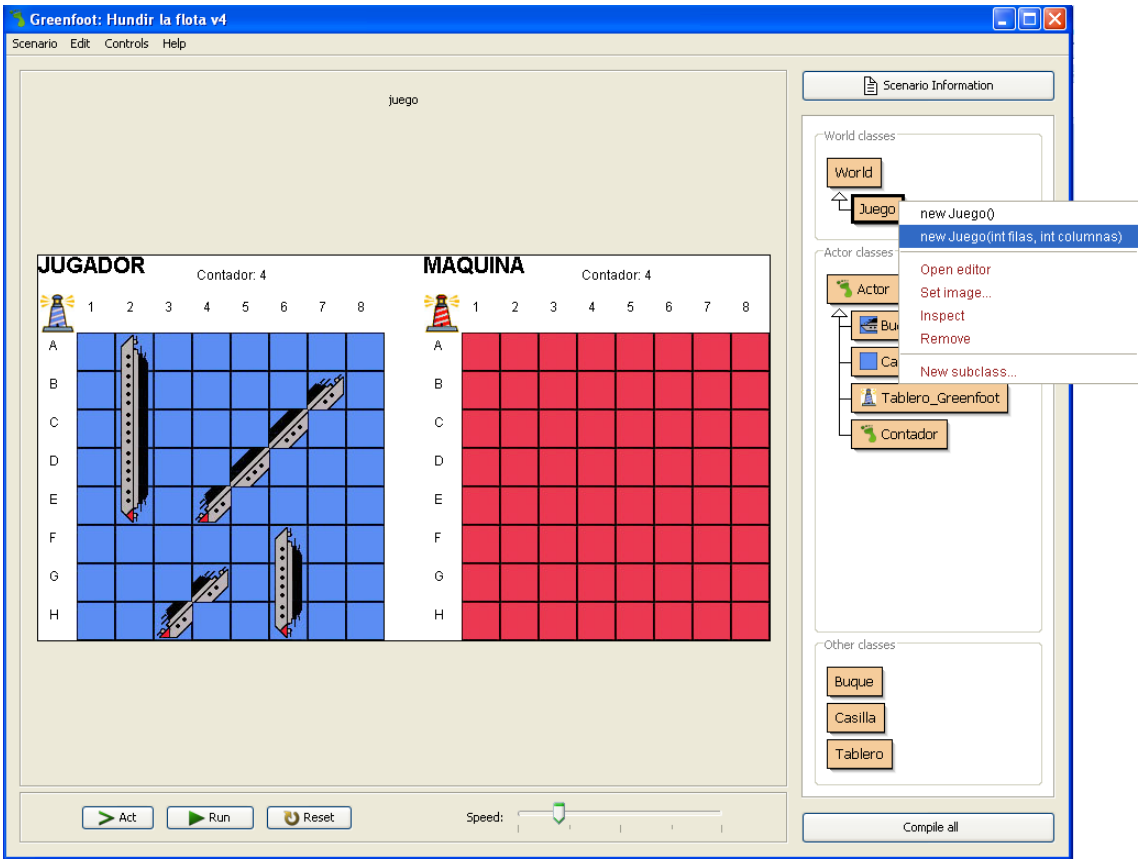


Figura 27. Creación de un juego con dimensiones NxM

En esta pantalla (Figura 27) se muestra como el usuario crea una instancia del juego en la que define las dimensiones de los tableros.

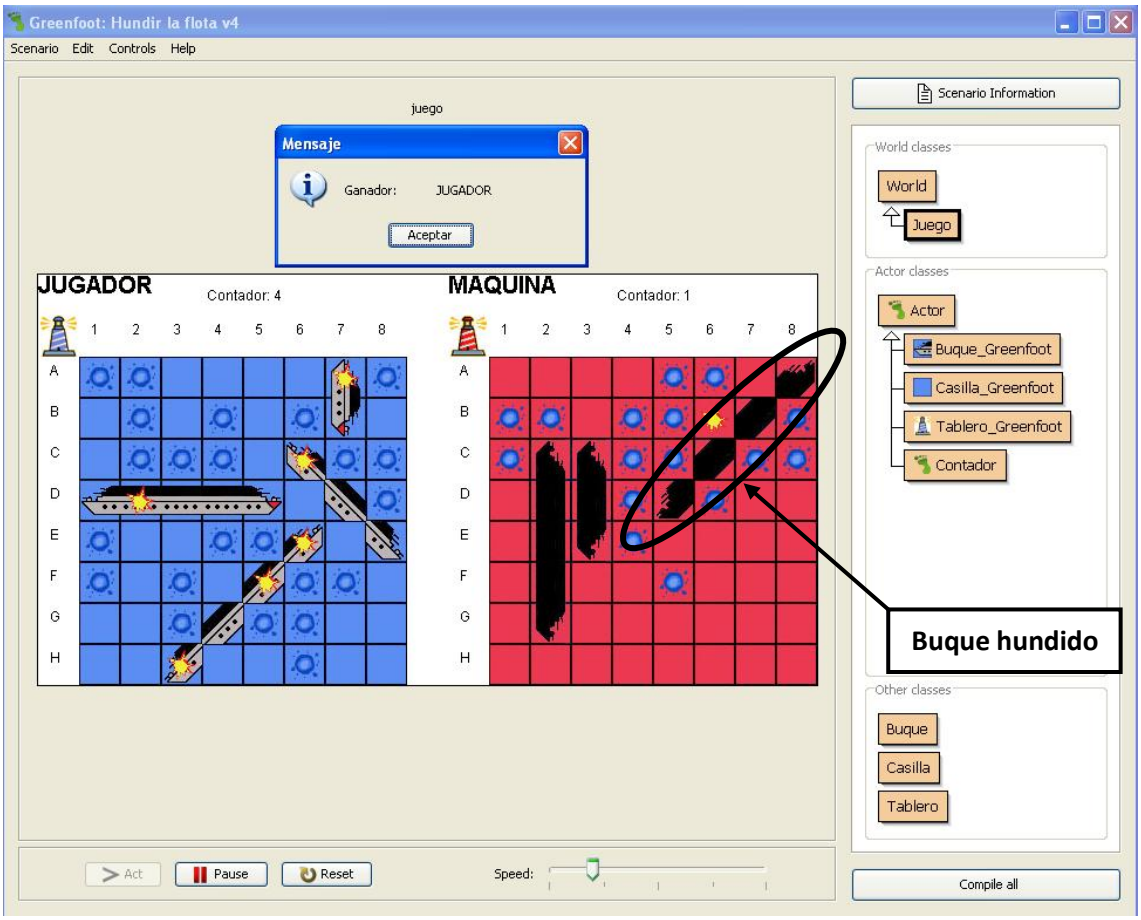


Figura 28. Fin del juego

Se muestra en esta pantalla (Figura 28), el resultado final de la ejecución del juego, en la que se muestran los buques hundidos, representados con las imágenes 'negras' de los buques.

8. Pruebas

Una vez terminado el proceso de implementación del proyecto, en este apartado se describirá las pruebas llevadas a cabo.

Los tipos de prueba se dividen de manera general en pruebas de verificación y validación. Las pruebas de verificación revisan si el resultado corresponde a la especificación del sistema, es decir si se está construyendo el sistema de manera correcta, algo que no garantiza la satisfacción del usuario final. Estas son las pruebas que se llevarán a cabo en este apartado.

Dentro de los tipos de pruebas existentes se han seleccionado como más adecuadas para este proyecto las llamadas pruebas basadas en requisitos o pruebas de casos de uso, con las que se intenta llevar a cabo pruebas basadas directamente en la especificación de requisitos. Se trata de verificar que el sistema final cumple con las especificaciones funcionales descritas por los casos de uso.

Para ello se han llevado a cabo dos niveles de pruebas diferentes: pruebas unitarias (PRU-U-XX) que tienen como objetivo la verificación del correcto funcionamiento de cada método y función, de manera que cada prueba será lo más independiente posible de las demás; y pruebas de sistema (PRU-S-XX) que tienen como objetivo la verificación del sistema completo o su aplicación como tal. Se toma el punto de vista del usuario final y los casos de prueba ejecutan acciones típicas del usuario.

8.1. Pruebas unitarias

Se realizan en este apartado las pruebas unitarias que tienen como objetivo la verificación del correcto funcionamiento de cada método y función. Dada la cantidad de pruebas unitarias, se ha decidido no definir específicamente en el documento cada una de ellas llevadas a cabo, sino que se mostrarán en detalle aquellas consideradas las más representativas.

Id	PRU-U-01
Clase	Casilla_Greenfoot
Método	bombardear()
Descripción	Bombardea una casilla.
Clases y métodos involucrados	Casilla_Greenfoot ▪ tieneBomba() ▪ tieneBuque() ▪ setImagenTocado(Casilla_Greenfoot c, Buque_Greenfoot b, int i) ▪ setImagenHundido(Casilla_Greenfoot c, Buque_Greenfoot b, int i) ▪ getCasilla() Casilla ▪ bombardear() Buque_Greenfoot ▪ hundido() Greenfoot ▪ playSound(String file)
Procedimiento	- Se comprueba si la casilla ha sido bombardeada anteriormente. - En caso de no haber sido bombardeada, se invoca al método bombardear() de la clase casilla, para bombardear la casilla. - Se comprueba si tiene buque: - En caso de tener buque se comprueba si solo ha sido tocado o si se ha hundido y se establece la imagen y se

	reproduce el .wav correspondiente. - En caso de no tener buque se establece la imagen del agua para indicarlo y se reproduce el .wav correspondiente.
Verificación	El valor de retorno debe ser TRUE si la casilla ha sido bombardeada.

Id	PRU-U-02
Clase	Buque_Greenfoot
Método	posicionar(Tablero_Greenfoot t, char fPopa, int cPopa, int iF, int iC)
Descripción	Posiciona el buque en el tablero dado con la popa en la casilla correspondiente a la fila y columna dada.
Clases y métodos involucrados	Buque ▪ posicionar(Tablero_Greenfoot t, char fPopa, int cPopa, int iF, int iC) ▪ getIntOrientacion(String o) Tablero_Greenfoot ▪ getCasilla(char f, int col) Casilla_Greenfoot ▪ alojar() Buque_Greenfoot ▪ setImagenCasilla(Tablero_Greenfoot t, Casilla_Greenfoot c, int i)
Procedimiento	- Se invoca al método posicionar de la clase Buque. - Se obtiene la orientación del buque para establecer los incrementos de la fila y la columna.

	- Se van obteniendo las casillas que pertenecerán al buque. - Se aloja el buque en la casilla correspondiente. - Se establece la imagen de la casilla.
Verificación	El valor de retorno debe ser TRUE si el buque ha sido posicionado.

Id	PRU-U-03
Clase	Tablero_Greenfoot
Método	bombardearCasilla(char fila, int co)
Descripción	Bombardea la casilla dada por la fila y la columna.
Clases y métodos involucrados	Tablero_Greenfoot ▪ enLimite(char fila, int columna) ▪ getCasilla(char fila, int col) Casilla_Greenfoot ▪ tieneBomba() ▪ bombardear()
Procedimiento	- Se comprueba si la casilla se encuentra dentro de los límites del tablero. - Se obtiene la casilla a bombardear. - Se comprueba si la casilla ha sido bombardeada anteriormente. - En caso de no haber sido bombardeada anteriormente, se bombardea la casilla.
Verificación	El valor de retorno debe ser TRUE si la casilla ha sido bombardeada.

Id	PRU-U-04
Clase	Tablero_Greenfoot
Método	ponerBuque(Buque_Greenfoot b, char fPopa, int cPopa)
Descripción	Posiciona el buque dado en el tablero con la popa en la fila correspondiente a la columna y la fila dada.
Clases y métodos involucrados	Tablero_Greenfoot ▪ enLimite(char fila, int columna) ▪ orientacionValida(int o) Buque_Greenfoot ▪ posicionar(Tablero_Greenfoot t, char fPopa, int cPopa, int iF, int iC) Tablero ▪ getArrayBuques() ▪ setSigPosBuque(int pos)
Procedimiento	- Se comprueba que la casilla en la que se quiere situar la popa del buque se encuentra dentro de los límites del tablero. - Se comprueba que la orientación del buque está dentro de las orientaciones consideradas como válidas. - Se posiciona el buque en el tablero, siempre y cuando no se haya superado ya todos los buques en el tablero.
Verificación	El valor de retorno debe ser TRUE si el buque ha sido posicionado en el tablero.

Id	PRU-U-05
Clase	Juego

Método	jugar()
Descripción	Simulación de un juego. Se bombardeará alternando turnos, el tablero del jugador y el de la máquina. La casilla a bombardear será elegida de manera aleatoria.
Clases y métodos involucrados	Greenfoot ▪ setSpeed(int speed) ▪ getRandomNumber(int limite) ▪ delay(int delay) Tablero_Greenfoot ▪ todosHundidos() ▪ numAFila(int f) ▪ getTableroJugador() ▪ getTableroMaquina() ▪ getCasilla() Casilla_Greenfoot ▪ bombardear() World ▪ repaint()
Procedimiento	- Se establece la velocidad de ejecución del juego. - Se comprueba si no están todos los buques del tablero del jugador hundidos. - Se obtiene una casilla con una fila y una columna aleatoria y se intenta bombardear. - Se repinta el tablero para mostrar las imágenes. - Se comprueba si no están todos los buques del tablero de la máquina hundidos.

	- Se obtiene una casilla con una fila y una columna aleatoria y se intenta bombardear. - Se repinta el tablero para mostrar las imágenes. - Se repite este proceso hasta fin del juego (uno de los dos tableros tenga todos los buques hundidos)
Verificación	Deben aparecer las casillas bombardeadas y en un determinado momento terminar el juego.

8.2. Pruebas de sistema

Se realizan en este apartado las pruebas de sistema que tienen como objetivo la verificación del sistema completo o su aplicación como tal, comprobando la funcionalidad a través de los casos de uso.

Se han realizado en algunos casos más de un caso de prueba por caso de uso, debido a que en ocasiones el mismo caso de uso puede ser llevado a cabo de distintas maneras (con la invocación de métodos distintos). Además se ha probado el escenario básico con parámetros no validos con la intención de comprobar que produce el error que debería producir.

Todas las pruebas han sido llevadas a cabo desde el punto de vista del usuario. Sin embargo debemos tener en cuenta que tal y como se especificó en el diagrama de casos de uso, existen algunos de ellos que pueden llevados a cabo internamente debido a que unos casos de uso incluyen otros.

<table>
<tr><td>Id</td><td colspan="2">PRU-S-01</td></tr>
<tr><td>Descripción</td><td colspan="2">Prueba asociada a la creación de una casilla</td></tr>
<tr><td rowspan="4">Caso de uso</td><td>Id</td><td>CU-03</td></tr>
<tr><td colspan="2">Descripción</td></tr>
<tr><td colspan="2">Crea una casilla que pertenecerá a un tablero</td></tr>
<tr><td colspan="2">Escenario básico</td></tr>
<tr><td>Secuencia de invocación</td><td colspan="2">En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Casilla el cual creará la casilla.</td></tr>
<tr><td>Procedimiento</td><td colspan="2">- Casilla_Greenfoot()

- Casilla()</td></tr>
<tr><td>Verificación</td><td colspan="2">Se selecciona en el menú de clases la clase Casilla_Greenfoot.

Se pincha con el botón derecho y se selecciona el constructor de la clase. Se pincha en el mundo creado y la casilla debe ser creada automáticamente.</td></tr>
<tr><td></td><td colspan="2">Existe la clase Casilla_Greenfoot.

Al pulsar el botón derecho sobre dicha clase aparece el constructor.

Tras seleccionar el constructor y pinchar en el mundo, se crea la casilla.

(Véase Figura 29)</td></tr>
</table>

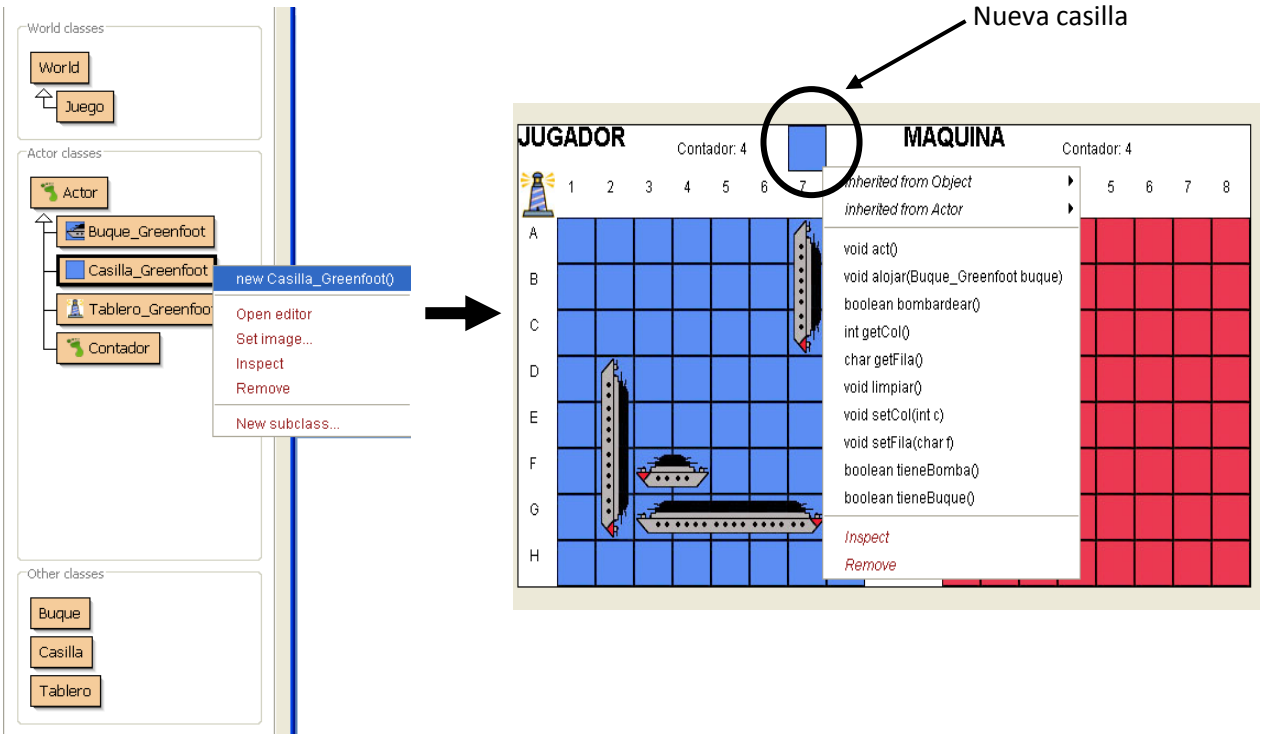


Figura 29. PRU-S-01

Id	PRU-S-02	
Descripción	Prueba asociada a la creación de un buque con una longitud mayor o igual a 2.	
Caso de uso	**Id**	CU-04
	Descripción	
	Crea un buque que podrá ser posicionado en el tablero.	
	Escenario	
	En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Buque que recibirá como parámetros el nombre, la longitud y la orientación de dicho buque y lo creará en el mundo.	
Secuencia de invocación	-Buque_Greenfoot(String nombre,int longitud,String orientación) -Buque (String nombre,int longitud,String orientación)	

Procedimiento	Se selecciona en el menú de clases la clase Buque_Greenfoot. Se pincha con el botón derecho y se selecciona el constructor de la clase. Se pincha en el mundo creado y el buque debe ser creado automáticamente.
Verificación	Existe la clase Buque_Greenfoot. Al pulsar el botón derecho sobre dicha clase aparece el constructor. Tras seleccionar el constructor y pinchar en el mundo, se crea el buque. (Véase Figura 30)

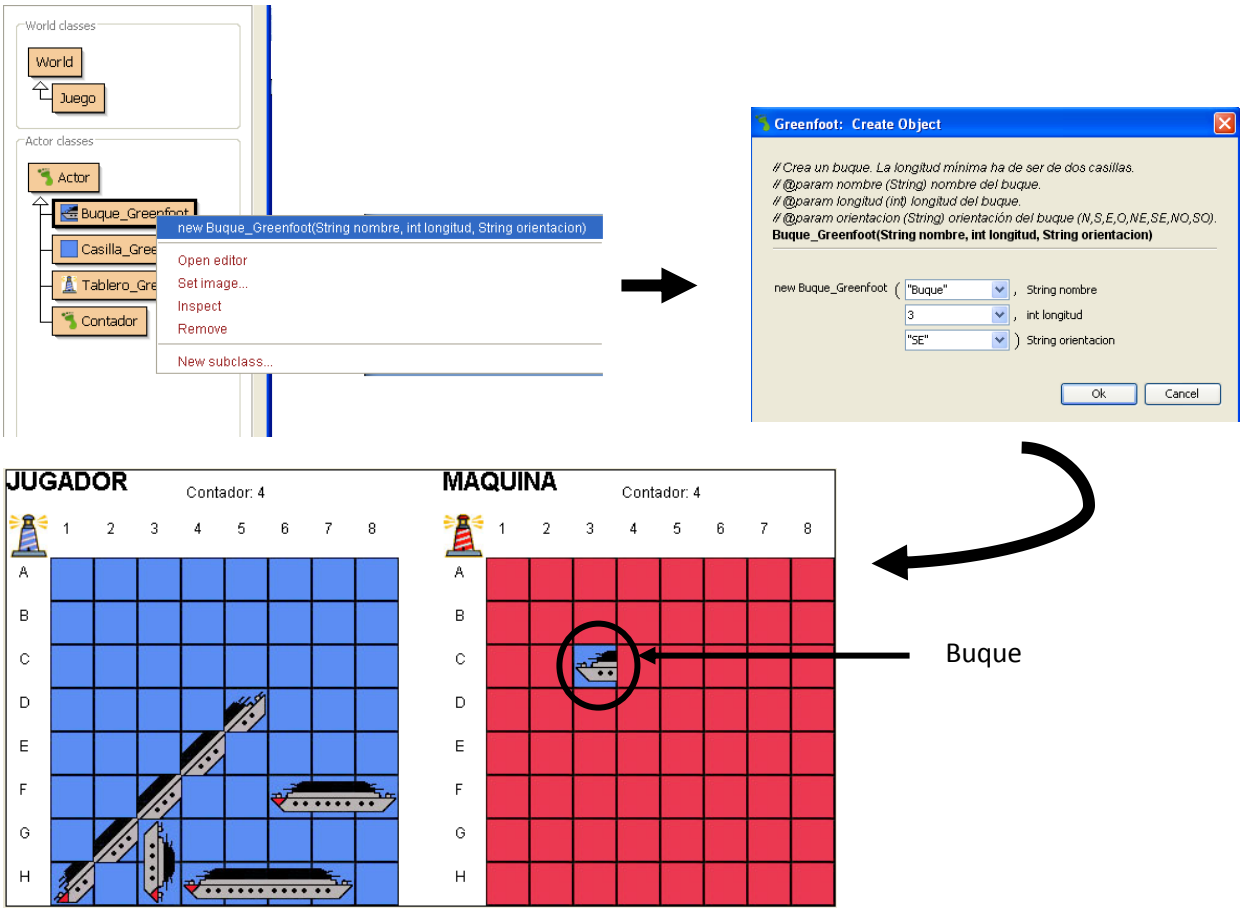


Figura 30. PRU-S-02

<table>
<tr><td>Id</td><td>PRU-S-03</td></tr>
<tr><td>Descripción</td><td>Prueba asociada a la creación de un buque con una longitud igual a 1.</td></tr>
<tr><td>Caso de uso</td><td>Id CU-04
Descripción
Crea un buque que podrá ser posicionado en el tablero.
Escenario
En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Buque que recibirá como parámetros el nombre, la longitud y la orientación de dicho buque y lo creará en el mundo.</td></tr>
<tr><td>Secuencia de invocación</td><td>-Buque_Greenfoot(String nombre,int longitud,String orientación)</td></tr>
<tr><td>Procedimiento</td><td>Se selecciona en el menú de clases la clase Buque_Greenfoot.

Se pincha con el botón derecho y se selecciona el constructor de la clase. Se pincha en el mundo creado y tras introducir el parámetro erróneo deberá aparecer un mensaje de alerta indicando que el buque no ha sido creado.</td></tr>
<tr><td>Verificación</td><td>Existe la clase Buque_Greenfoot.

Al pulsar el botón derecho sobre dicha clase aparece el constructor.

Tras seleccionar el constructor y pinchar en el mundo, se introducen la longitud con valor 1. Tras darle pinchar el botón Aceptar, aparece un mensaje de alerta indicando que el buque no ha sido creado.

(Véase Figura 31)</td></tr>
</table>

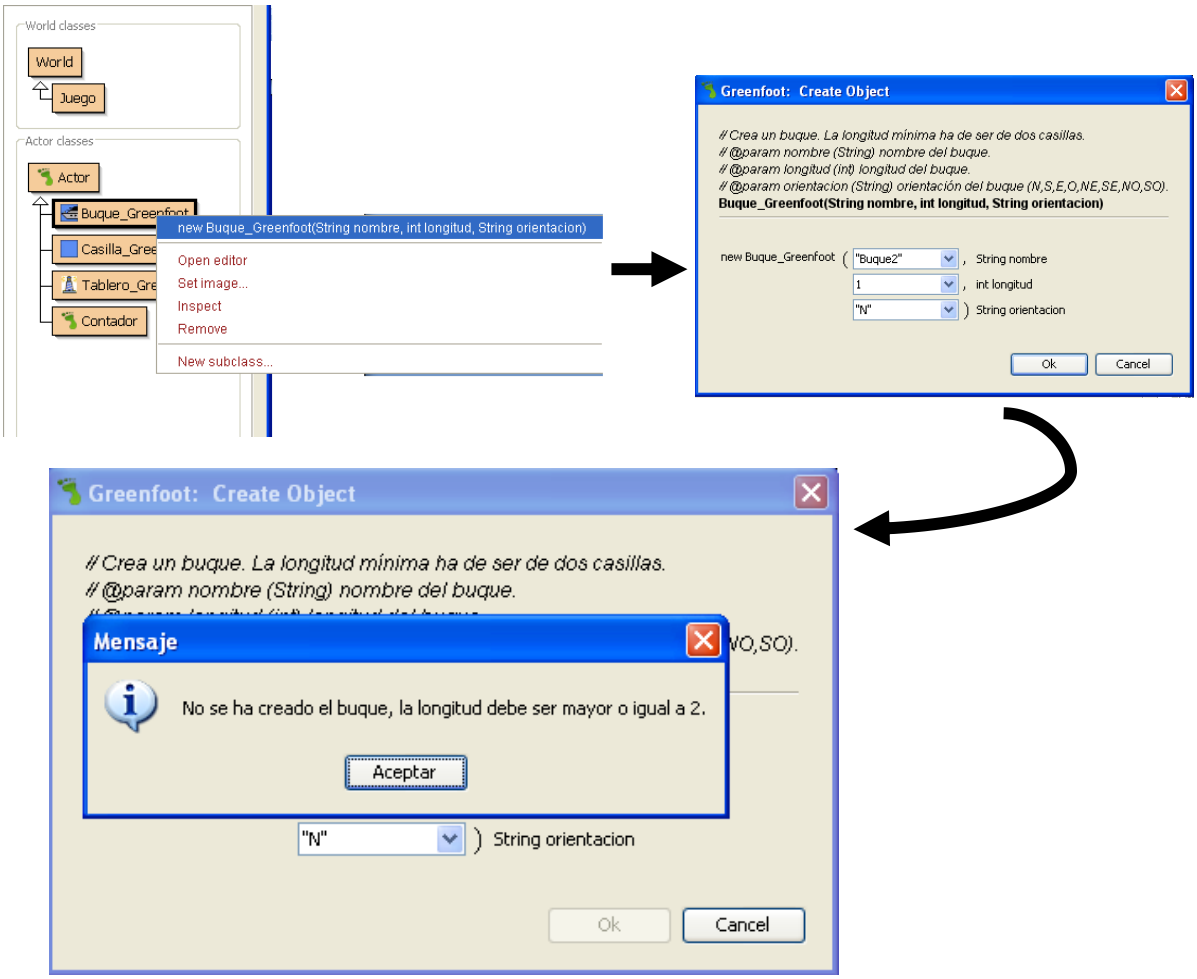


Figura 31. PRU-S-03

Id	PRU-S-04	
Descripción	Prueba asociada a la creación de un tablero con valores entre 8 y 20	
Caso de uso	**Id**	CU-02
	Descripción	
	Crear los tableros que formarán el juego. Un tablero para el jugador y otro para la máquina.	
	Escenario	
	En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Tablero que creará	

	un array bidimensional de casillas.
Procedimiento	Se selecciona en el menú de clases la clase Tablero_Greenfoot. Se pincha con el botón derecho y se selecciona el constructor de la clase. Se pincha en el mundo creado y el tablero debe ser creado automáticamente.
Secuencia de invocación	-Tablero_Greenfoot(int filas,int columnas, int numBuques) - Tablero(int filas,int columnas, int numBuques) -Casilla_Greenfoot() -Contador()
Verificación	Existe la clase Tablero_Greenfoot. Al pulsar el botón derecho sobre dicha clase aparece el constructor. Tras seleccionar el constructor y pinchar en el mundo, se crea el tablero. (Véase Figura 32)

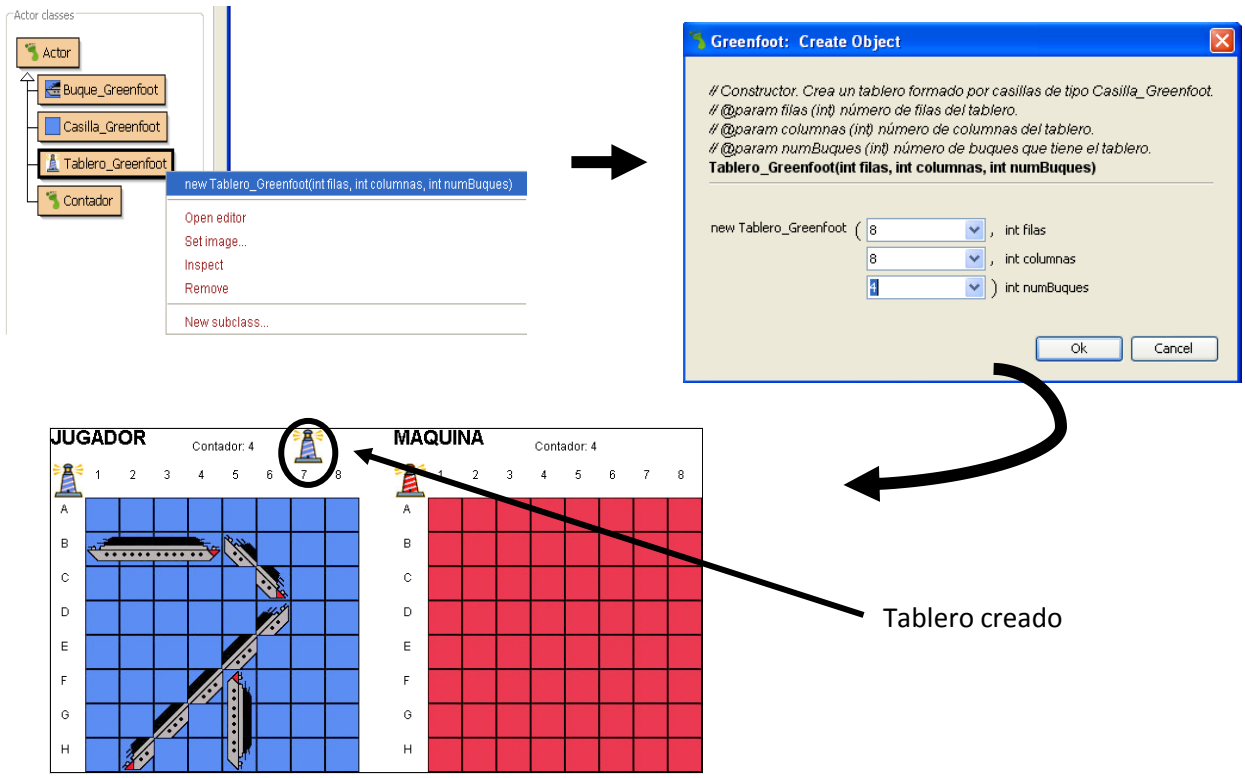


Figura 32. PRU-S-04

Id	PRU-S-05	
Descripción	Prueba asociada a la creación de un tablero con valores fuera de los límites (mínimo: 8, máximo: 20)	
Caso de uso	**Id**	CU-02
	Descripción	
	Crear los tableros que formarán el juego. Un tablero para el jugador y otro para la máquina.	
	Escenario	
	En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Tablero que creará un array bidimensional de casillas.	
Secuencia de invocación	-Tablero_Greenfoot(int filas,int columnas, int numBuques)	

Procedimiento	Se selecciona en el menú de clases la clase Tablero_Greenfoot. Se pincha con el botón derecho y se selecciona el constructor de la clase. Se pincha en el mundo creado y se introducirán los parámetros fuera de los límites. Tras pulsar el botón aceptar aparecerá un mensaje de alerta indicando que el tablero no ha sido creado.
Verificación	Existe la clase Tablero_Greenfoot. Al pulsar el botón derecho sobre dicha clase aparece el constructor. Tras seleccionar el constructor y pinchar en el mundo, se introducen los parámetros no válidos y aparece un mensaje de alerta indicando que el tablero no ha sido creado. (Véase Figura 33)

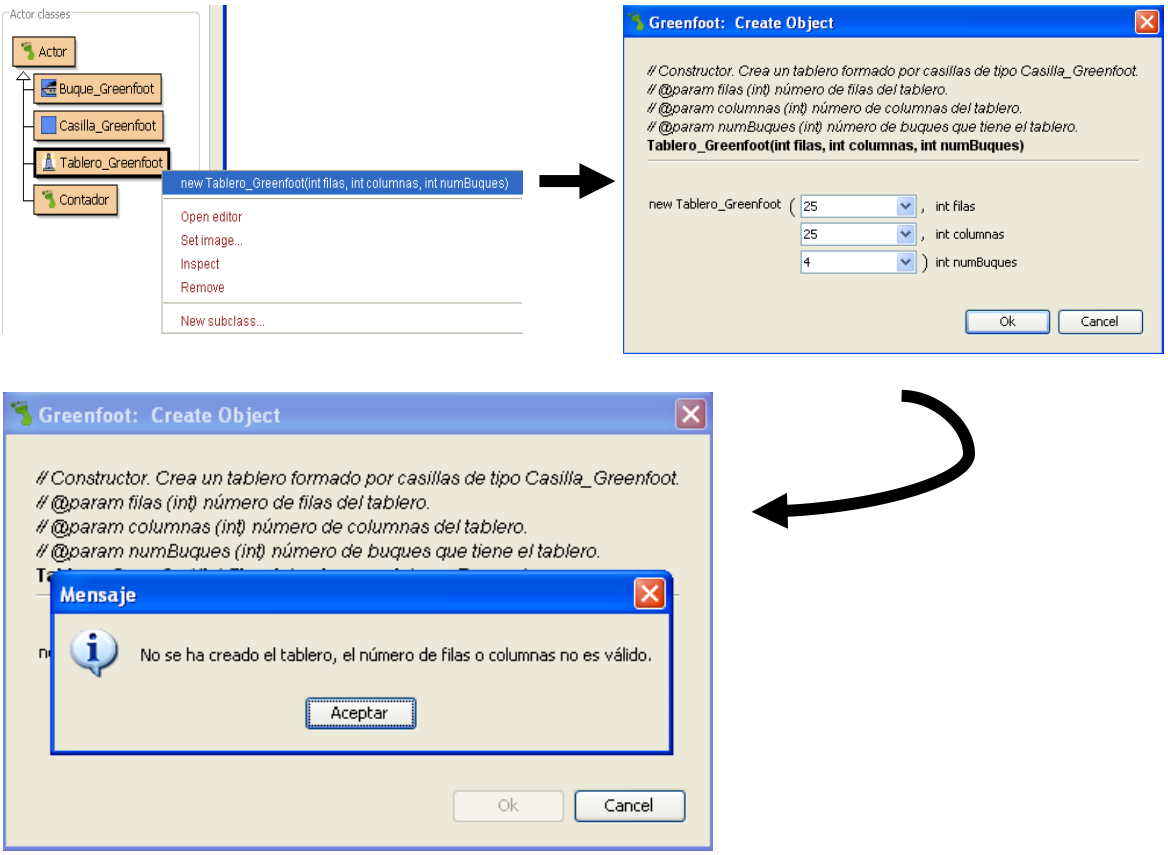


Figura 33. PRU-S-05

<table>
<tr><td>Id</td><td colspan="2">PRU-S-06</td></tr>
<tr><td>Descripción</td><td colspan="2">Prueba asociada a la creación de un juego por defecto (8x8)</td></tr>
<tr><td rowspan="4">Caso de uso</td><td>Id</td><td>CU-01</td></tr>
<tr><td colspan="2">Descripción</td></tr>
<tr><td colspan="2">La creación de un nuevo juego, bien sea un juego por defecto con el tamaño de los tableros 8x8 o un juego en el que el usuario decida el número de filas y casillas de los tableros.</td></tr>
<tr><td colspan="2">Escenario</td></tr>
<tr><td></td><td colspan="2">En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Juego el cual creará dos tableros, uno para el jugador y otro para la máquina y posicionará cuatro buques en cada uno de los tableros, al azar.</td></tr>
<tr><td>Secuencia de invocación</td><td colspan="2">-iniciarJuego()

- Tablero_Greenfoot(int filas,int columnas, int numBuques)

-Tablero_Greenfoot(int filas,int columnas, int numBuques)

-pintarTablero()

-posicionarBuquesAzar()

-posicionarBuquesAzar()</td></tr>
<tr><td>Procedimiento</td><td colspan="2">Se pincha con el botón derecho sobre la etiqueta juego y se selecciona el método ‘iniciarJuego’ que creará un tablero de tamaño 8x8.</td></tr>
<tr><td>Verificación</td><td colspan="2">Existe la etiqueta juego.

Al pulsar el botón derecho sobre dicha etiqueta aparece los el método correspondiente.

Tras seleccionarlo se pinta automáticamente.

(Véase Figura 34)</td></tr>
</table>

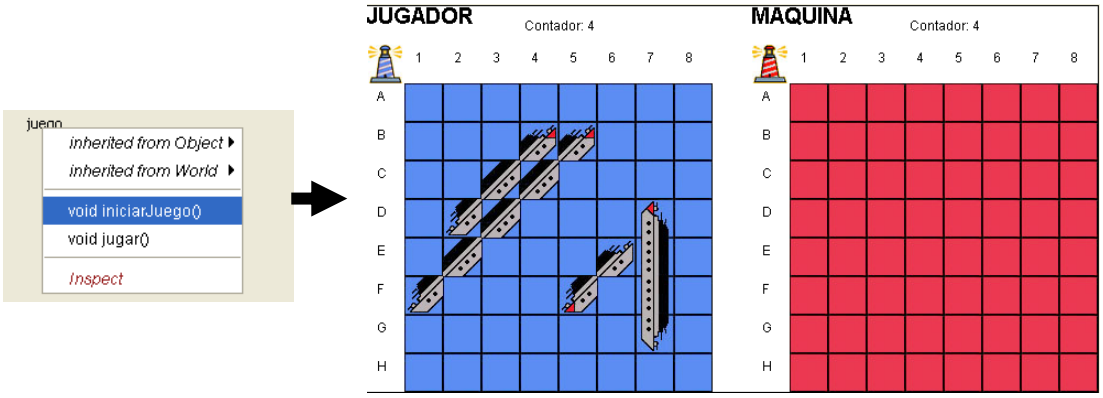


Figura 34.PRU-S-06

Id	PRU-S-07	
Descripción	Prueba asociada a la creación de un juego con parámetros válidos (filas y columnas dentro de los límites)	
Caso de uso	**Id**	CU-01
	Descripción	
	La creación de un nuevo juego, bien sea un juego por defecto con el tamaño de los tableros 8x8 o un juego en el que el usuario decida el número de filas y casillas de los tableros.	
	Escenario	
	En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Juego el cual creará dos tableros, uno para el jugador y otro para la máquina y posicionará cuatro buques en cada uno de los tableros, al azar.	
Secuencia de invocación	-Juego (int filas, int columnas) - Tablero_Greenfoot(int filas,int columnas, int numBuques) -Tablero_Greenfoot(int filas,int columnas, int numBuques)	

	-pintarTablero() -posicionarBuquesAzar() -posicionarBuquesAzar()
Procedimiento	Se selecciona en el menú de clases la clase Juego. Se pincha con el botón derecho y se selecciona el constructor de la clase que se desee, en este caso el de por defecto "Juego(filas,columnas)", que creará un tablero de tamaño NxM. En este caso no es necesario pulsar en el mundo para crear el juego.
Verificación	Existe la clase Tablero_Greenfoot. Al pulsar el botón derecho sobre dicha clase aparecen los constructores de la clase Juego. Tras seleccionar el constructor e introducir los parámetros válidos se crea el juego. (Véase Figura 35)

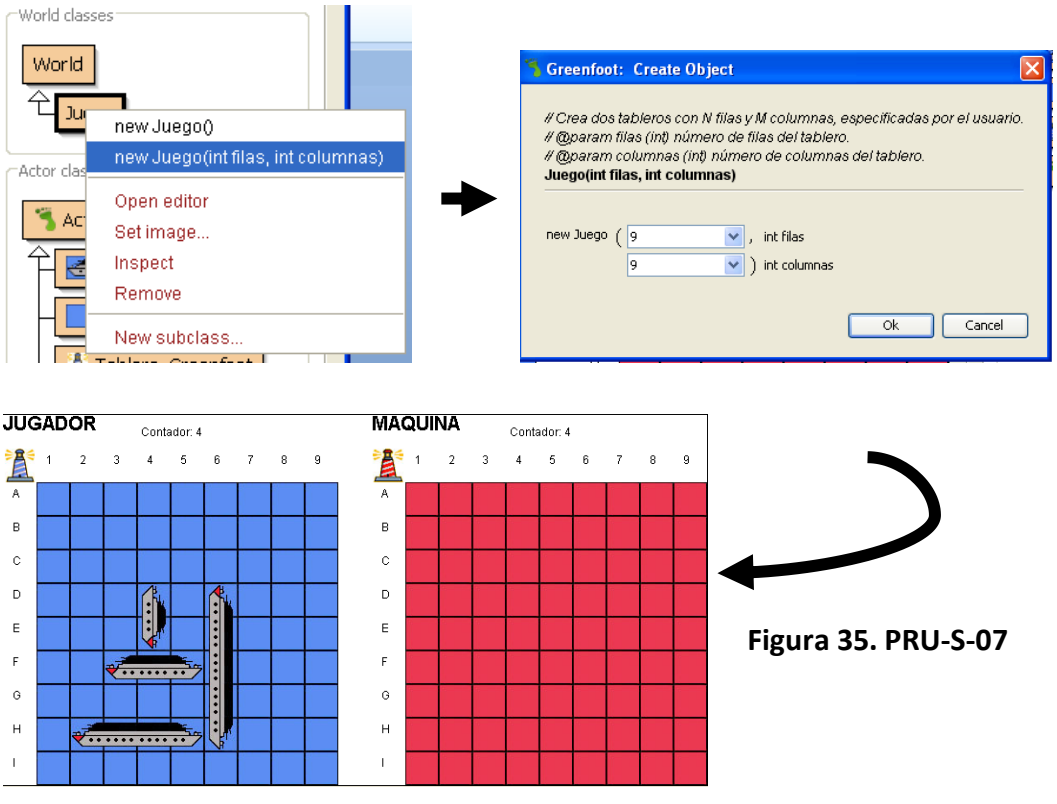


Figura 35. PRU-S-07

<table>
<tr><td>Id</td><td colspan="2">PRU-S-08</td></tr>
<tr><td>Descripción</td><td colspan="2">Prueba asociada a la creación de un juego con parámetros no válidos (filas y columnas fuera de los límites)</td></tr>
<tr><td rowspan="5">Caso de uso</td><td>Id</td><td>CU-01</td></tr>
<tr><td colspan="2">Descripción</td></tr>
<tr><td colspan="2">La creación de un nuevo juego, bien sea un juego por defecto con el tamaño de los tableros 8x8 o un juego en el que el usuario decida el número de filas y casillas de los tableros.</td></tr>
<tr><td colspan="2">Escenario</td></tr>
<tr><td colspan="2">En el apartado correspondiente a las clases se realizará la invocación del constructor de la clase Juego el cual creará dos tableros, uno para el jugador y otro para la máquina y posicionará cuatro buques en cada uno de los tableros, al azar.</td></tr>
<tr><td>Secuencia de invocación</td><td colspan="2">-Juego (int filas, int columnas)</td></tr>
<tr><td>Procedimiento</td><td colspan="2">Se selecciona en el menú de clases la clase Juego.

Se pincha con el botón derecho y se selecciona el constructor de la clase que se desee, en este caso el de por defecto “Juego(filas,columnas)”, que creará un tablero de tamaño NxM. En este caso no es necesario pulsar en el mundo para crear el juego.

Deberá aparecer un mensaje de alerta.</td></tr>
<tr><td>Verificación</td><td colspan="2">Existe la clase Tablero_Greenfoot.

Al pulsar el botón derecho sobre dicha clase aparecen los constructores de la clase Juego.

Tras seleccionar el constructor e introducir los parámetros no válidos aparece un mensaje de alerta.</td></tr>
</table>

	(Véase Figura 36)

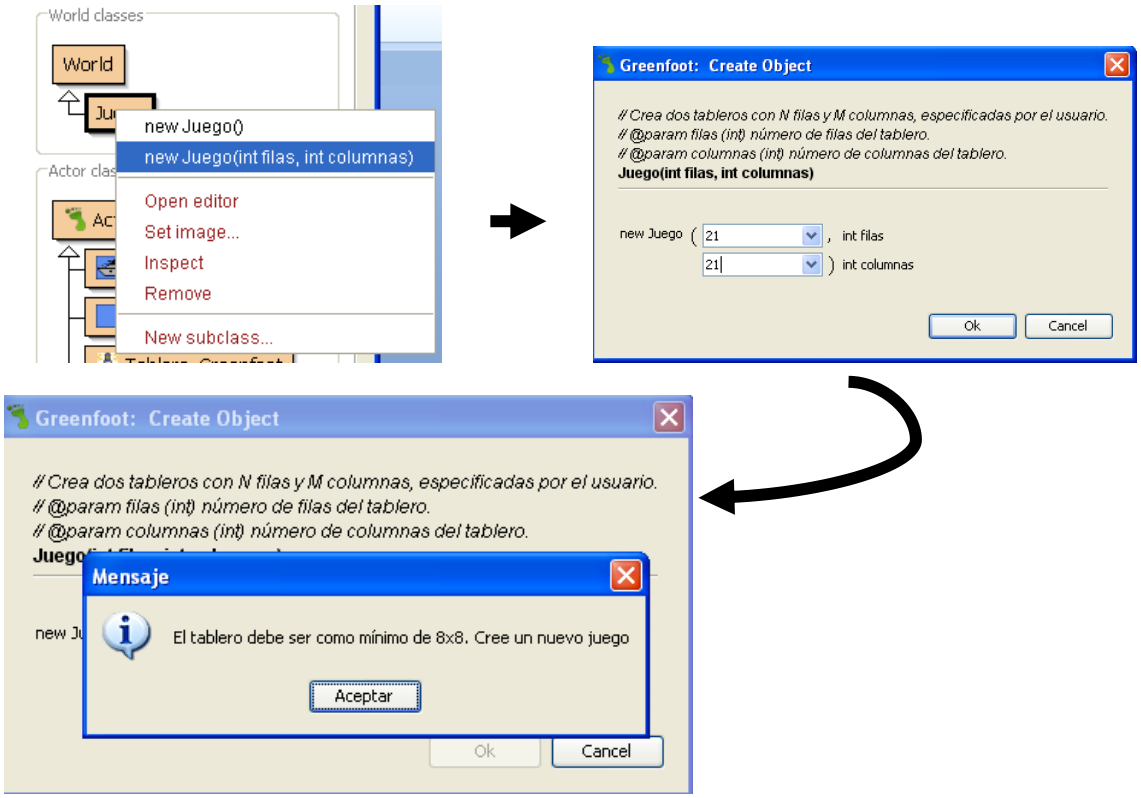


Figura 36. PRU-S-08

Id	PRU-S-09	
Descripción	Prueba asociada al posicionamiento de un buque en un tablero	
Caso de uso	**Id**	CU-05
	Descripción	
	Posiciona un buque en el tablero.	
	Escenario	
	En primer lugar se comprobará si las casillas a ocupar por el buque se encuentran dentro de los límites del tablero, y si la orientación del buque es válida.	

	En caso de ser verdaderas las dos condiciones anteriores, se comprobará en segundo lugar si han sido posicionados en el tablero todos los buques. Después se comprobará si se puede posicionar en dichas casillas, para ello se procede a mirar si las casillas no contienen otro buque y si el citado buque no se cruza con otro buque posicionado anteriormente. Por último, siendo afirmativas también las condiciones anteriores, en la ejecución normal, los buques ya estarán posicionados por lo que deberá aparecer un mensaje de alerta.
Secuencia de invocación	-ponerBuque(Buque_Greenfoot b,char fPopa, int cPopa) -enLimite(char fila, int col) -orientacionValida(int o)
Procedimiento	Se pincha botón derecho en la imagen que representa el tablero (faro) y se selecciona el método “ponerBuque” el cual tiene como parámetros el buque que queremos posicionar, (para lo que se selecciona el buque y se añade directamente) y la fila y la columna.
Verificación	Existe la imagen correspondiente al tablero y por tanto el tablero creado previamente. Al pulsar botón derecho, en la lista de métodos públicos de la clase Tablero_Greenfoot está el método “ponerBuque”. Tras seleccionarlo sale la ventana correspondiente. Y al seleccionar el buque se añade automáticamente. Una vez rellenos los parámetros, aparece un mensaje de alerta indicando que el buque no ha podido ser añadido. (Véase Figura 37)

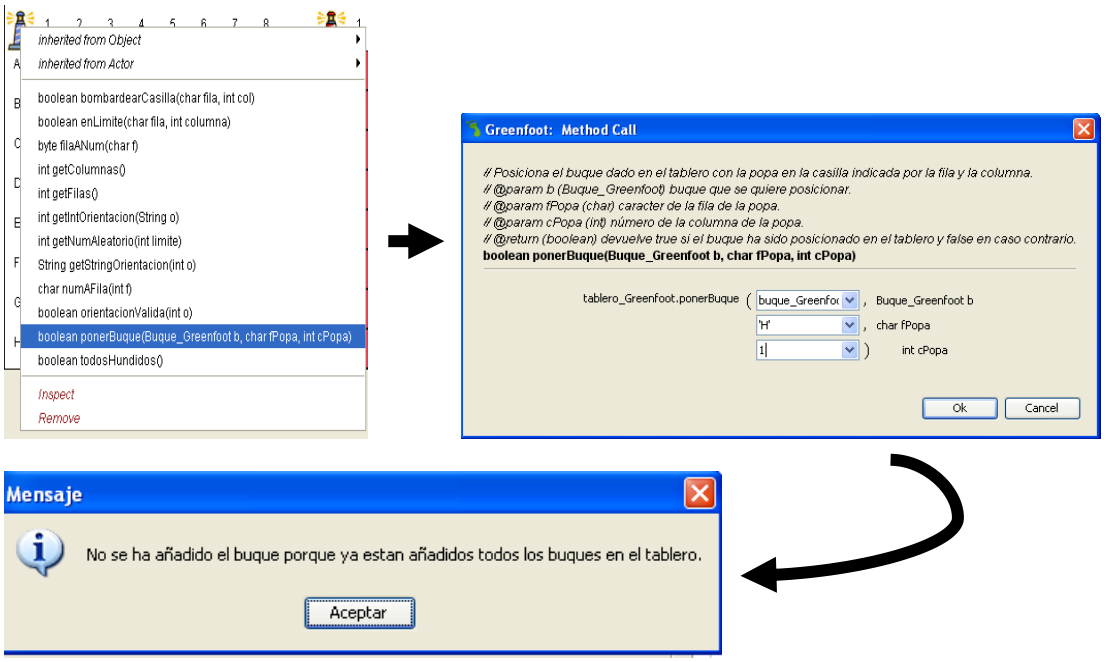


Figura 37. PRU-S-09

Id	PRU-S-10	
Descripción	Prueba asociada a bombardear una casilla de un tablero dentro de los límites	
Caso de uso	**Id**	CU-07
	Descripción	
	Bombardear una casilla para intentar hundir los buques del contrario.	
	Escenario	
	En primer lugar se comprobará si la casilla se encuentra dentro de los límites del tablero. En caso de que se encuentre en los límites del tablero, se comprobará que dicha casilla no ha sido bombardeada con anterioridad. Si dicha casilla no ha sido bombardeada anteriormente, se bombardeará la casilla.	

Secuencia de invocación	-bombardearCasilla(char fila, int col) -getColumnas() -enLimite(char fila, int col) -getCasilla(char fila, int col) -tieneBomba() -bombardear()
Procedimiento	Se pincha botón derecho en el tablero en el que se quiere bombardear. Al pulsar botón derecho, en la lista de métodos públicos de la clase Tablero_Greenfoot está el método "bombardearCasilla", que tiene como parámetros la fila y la columna de la casilla a bombardear. Se selecciona y la casilla será bombardeada siempre y cuando no haya sido bombardeada anteriormente.
Verificación	Existe la imagen del tablero y por tanto el tablero creado previamente. Tras pulsar botón derecho sobre dicha imagen aparece, entre los métodos públicos el método "bombardearCasilla". Una vez especificadas la fila y la columna se bombardea la casilla siempre que cumple las condiciones asociadas (siempre que no ha sido bombardeada anteriormente). (Véase Figura 38)

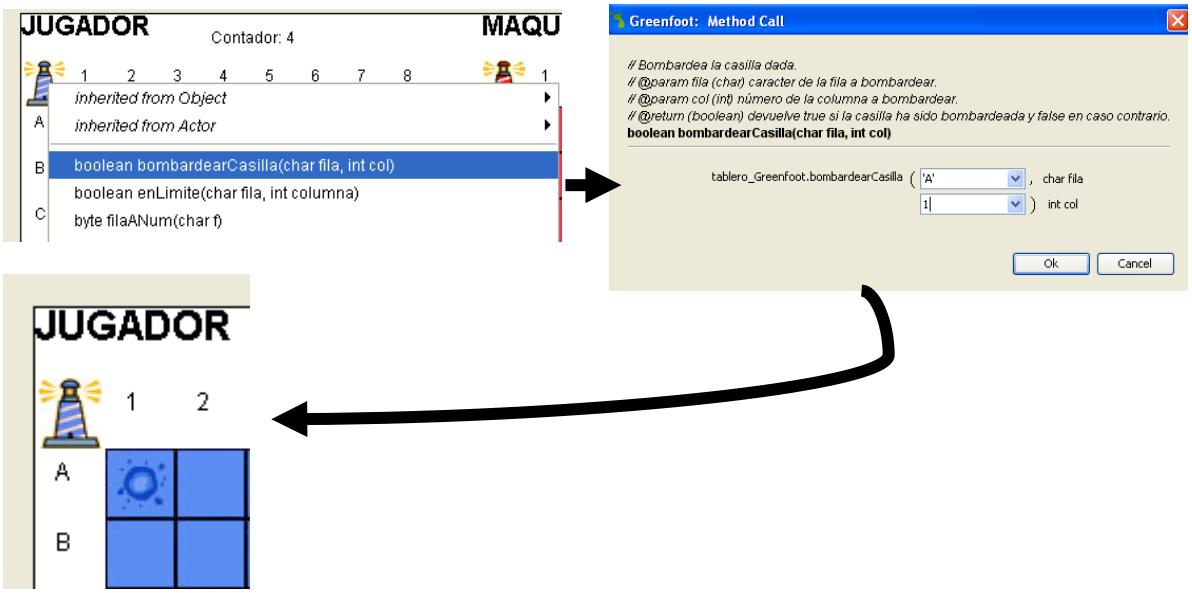


Figura 38. PRU-S-10

Id	PRU-S-11	
Descripción	Prueba asociada a bombardear una casilla de un tablero fuera de los límites	
Caso de uso	**Id**	CU-07
	Descripción	
	Bombardear una casilla para intentar hundir los buques del contrario.	
	Escenario	
	En primer lugar se comprobará si la casilla se encuentra dentro de los límites del tablero. En caso de que se encuentre en los límites del tablero, se comprobará que dicha casilla no ha sido bombardeada con anterioridad. Si dicha casilla no ha sido bombardeada anteriormente, se bombardeará la casilla.	
Secuencia de	-bombardearCasilla(char fila, int col)	

invocación	-getColumnas() -enLimite(char fila, int col)
Procedimiento	Se pincha botón derecho en el tablero en el que se quiere bombardear. Al pulsar botón derecho, en la lista de métodos públicos de la clase Tablero_Greenfoot está el método "bombardearCasilla", que tiene como parámetros la fila y la columna de la casilla a bombardear. Se selecciona y deberá aparecer una ventana de Greenfoot con valor false, indicando que la casilla no ha sido bombardeada.
Verificación	Existe la imagen del tablero y por tanto el tablero creado previamente. Tras pulsar botón derecho sobre dicha imagen aparece, entre los métodos públicos el método "bombardearCasilla". Una vez especificadas la fila y la columna aparece una ventana de Greenfoot con valor false. (Véase Figura 39)

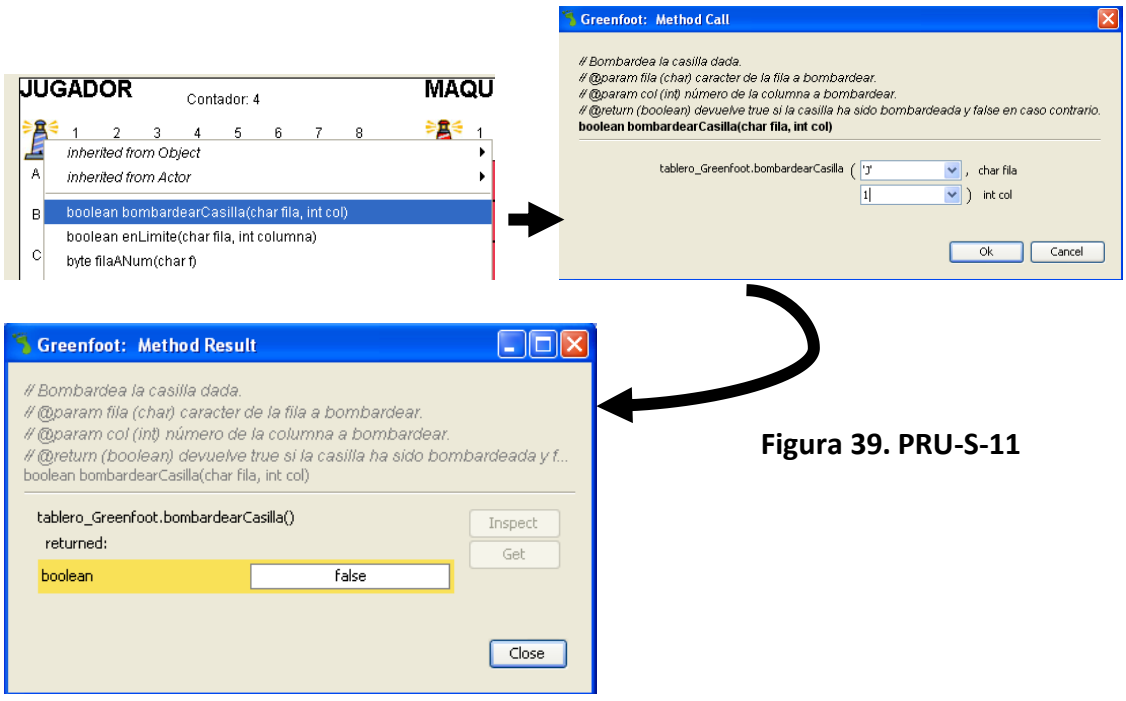


Figura 39. PRU-S-11

<table>
<tr><td>Id</td><td colspan="2">PRU-S-12</td></tr>
<tr><td>Descripción</td><td colspan="2">Prueba asociada al juego de “Hundir la flota”.</td></tr>
<tr><td rowspan="4">Caso de uso</td><td>Id</td><td>CU-06</td></tr>
<tr><td colspan="2">Descripción</td></tr>
<tr><td colspan="2">Jugar una partida al juego “Hundir la flota”.</td></tr>
<tr><td colspan="2">Escenario</td></tr>
<tr><td></td><td colspan="2">Jugar automáticamente: el usuario invocará al método jugar() el cual bombardeará aleatoriamente, alternando turno, primero en un tablero (el del jugador, por ejemplo) y luego en el otro, mostrando visualmente si se ha alcanzado un buque o no.</td></tr>
<tr><td>Secuencia de invocación</td><td colspan="2">-jugar()

-todosHundidos() (1)

-numAFila(int num) (2)

-getNumAleatorio(int limite) (3)

-bombardearCasilla(char fila, int col) (4)

(La secuencia que contiene los métodos 1,2,3 y 4 se repite tanto para el jugador como para la máquina, hasta el final del juego)</td></tr>
<tr><td>Procedimiento</td><td colspan="2">Se pincha botón derecho en la palabra “juego”, que aparece en la parte superior (encima de los tableros) que representa el mundo creado por la clase Juego, y se selecciona el método “jugar”.</td></tr>
<tr><td>Verificación</td><td colspan="2">Existe el juego creado previamente y el método “jugar”.

Tras seleccionar el método “jugar” se inicia el juego automático. Se bombardean la casillas del tablero del jugador y de la máquina alternándose, hasta que uno de los dos tableros tiene todos los buques hundidos, lo que supone el fin del juego.</td></tr>
</table>

	(En este caso no se ha considerado necesaria una imagen representativa)

Id	PRU-S-13	
Descripción	Prueba asociada al juego de “Hundir la flota”.	
Caso de uso	**Id**	CU-06
	Descripción	
	Jugar una partida al juego “Hundir la flota”.	
	Escenario	
	Jugar interactuando: el usuario pulsará el botón run y hará click, en el tablero de la máquina, en la casilla que quiera bombardear y automáticamente se bombardeará aleatoriamente el tablero de la máquina. En este caso también se mostrará visualmente si se ha alcanzado un buque o no.	
Procedimiento	Se pulsa el botón “Run” y se inicia así el juego. Se clickea en el tablero de la máquina la casilla que se quiere bombardear. Y así sucesivamente hasta que se finalice el juego o se quiera parar la ejecución del juego pulsando el botón “Pause”.	
Verificación	Tras pulsar el botón “Run” se inicia la ejecución y las casillas son bombardeadas al clickear una casilla que no ha sido bombardeada previamente. El juego finaliza una vez que uno de los dos tableros tiene todos los buques hundidos. (En este caso no se ha considerado necesaria una imagen representativa)	

8.3. Resultados

Los resultados obtenidos tras la realización de las pruebas unitarias planteadas para esta fase del proyecto se presentan en la siguiente tabla resumen (Tabla 7):

Id	Resultado	Descripción del fallo (si procede)
PRU-U-01	SATISFACTORIO	
PRU-U-02	SATISFACTORIO	
PRU-U-03	SATISFACTORIO	
PRU-U-04	SATISFACTORIO	
PRU-U-05	SATISFACTORIO	

Tabla 7. Resultados de las pruebas unitarias

Se presentan ahora los resultados obtenidos tras la realización de las pruebas de sistema planteadas para esta fase del proyecto en la siguiente tabla resumen (Tabla 8):

Id de la prueba	Resultado	Descripción del fallo (si procede)
PRU-S-01	SATISFACTORIO	
PRU-S-02	SATISFACTORIO	
PRU-S-03	SATISFACTORIO	
PRU-S-04	SATISFACTORIO	
PRU-S-05	SATISFACTORIO	
PRU-S-06	SATISFACTORIO	
PRU-S-07	SATISFACTORIO	
PRU-S-08	SATISFACTORIO	

PRU-S-09	SATISFACTORIO	
PRU-S-10	SATISFACTORIO	
PRU-S-11	SATISFACTORIO	
PRU-S-12	SATISFACTORIO	
PRU-S-13	SATISFACTORIO	

Tabla 8. Resultados de las pruebas de sistema

Los resultados obtenidos de las pruebas realizadas llevan a concluir que se ha alcanzado el correcto funcionamiento de la aplicación desarrollada, cubriendo todas las funcionalidades planteadas en los casos de uso. Se consigue por ello el objetivo de ofrecer una solución al problema planteado al inicio del documento. Sin embargo aún queda concluir si dicha solución proporcionará los objetivos planteados.

9. Conclusión

9.1. Aportaciones realizadas

Todo trabajo que se presente como Proyecto Fin de Carrera tiene un objetivo y por tanto unas aportaciones que podrán ser adquiridas a corto o largo plazo. Es por ello que se hace imprescindible concluir el documento describiendo las distintas aportaciones que se buscan realizar, en caso de no haber sido demostradas con total evidencia hasta el momento.

Las aportaciones que se pretenden realizar con este proyecto, planteado como Proyecto Fin de Carrera, son dos: (1) proporcionar un mecanismo que permita representar conceptos de la enseñanza de la programación orientada a objetos en Java y (2) la introducción de los usuarios en el uso de herramientas de apoyo.

Respecto a la primera aportación, se puede comenzar enunciando como la más valiosa aportación, proporcionar un mecanismo que permita comprender los conceptos de la enseñanza de la programación orientada a objetos en Java. La existencia de la problemática planteada en el inicio del documento y que ha desencadenado el desarrollo del presente proyecto ha permitido definir los principales objetivos que se han pretendido proporcionar.

El desarrollo del escenario del juego “Hundir la flota” que se ha explicado a lo largo de la documentación, junto con el uso de la herramienta Greenfoot también descrita en el Apartado 2.1, son los que nos proporcionan dicha aportación. El uso combinado del escenario con la herramienta Greenfoot, pretende proporcionar a los usuarios una comprensión de distintos y variados conceptos de la programación orientada a objetos, así como la sintaxis Java. Todo ello principalmente mediante la visualización de objetos (instanciación, estado, comportamiento..), pues es esta visualización la que permite fijar conceptos que normalmente se encuentran en la imaginación de los usuarios que comienzan en el mundo de la programación orientada a objetos y que no resulta tan intuitiva como la visualización.

Se muestra a continuación como se cumplen los objetivos planteados en el capítulo de introducción como consecuencia de esta primera aportación.

Diferencia entre clase y objeto

Tiene como prioridad diferenciar entre clases, objetos y constructores, conceptos que pueden ser confusos. Entender estas diferencias es importante para saber

cuando estamos diseñando un objeto y cuando estamos creando un objeto y cómo se construyen nuestros objetos.

Este objetivo ha sido cubierto en mayor medida por la selección de la herramienta considerada como la más adecuada, Greenfoot. Esta herramienta proporciona una visualización de los objetos creados a través de la invocación de los constructores, los cuales serán invocados desde la clase, lo que permite una mejor diferenciación entre los conceptos de clase y objeto.

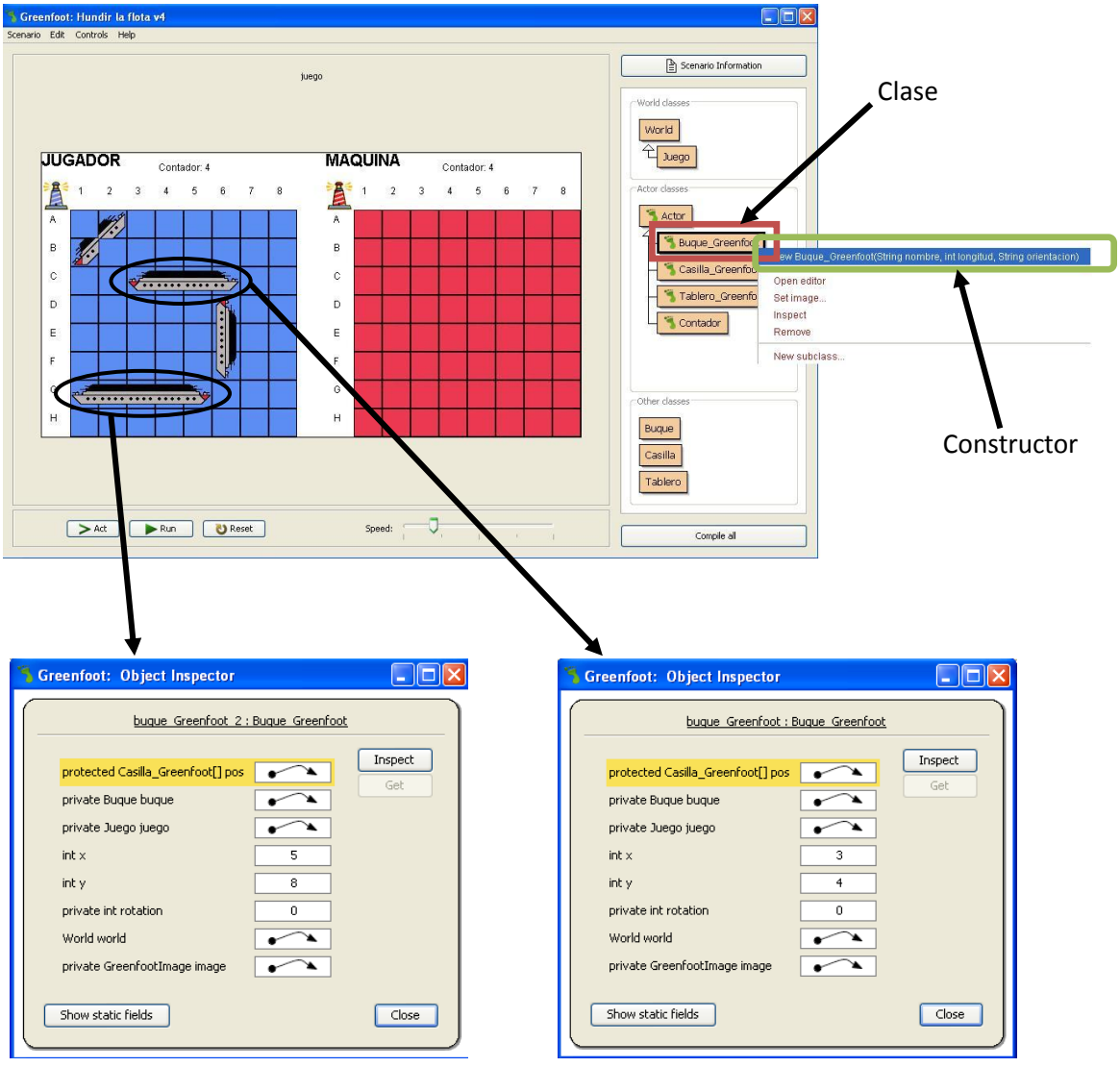


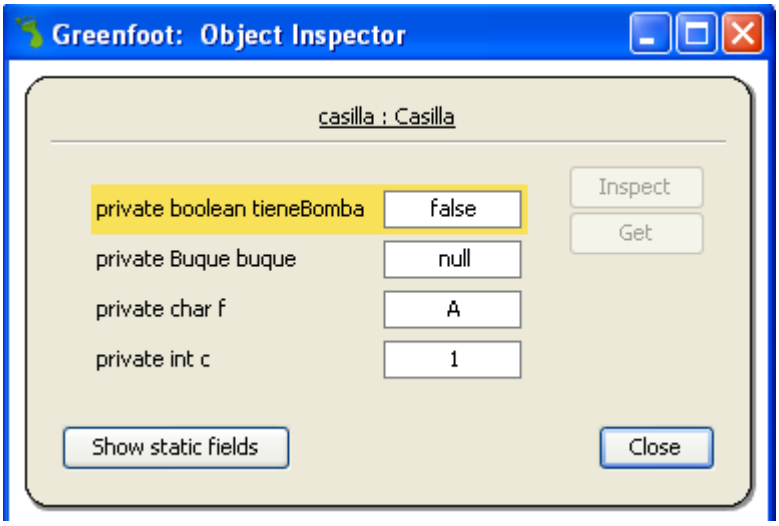
Figura 40. Objetivo: Diferencia entre clase y objeto

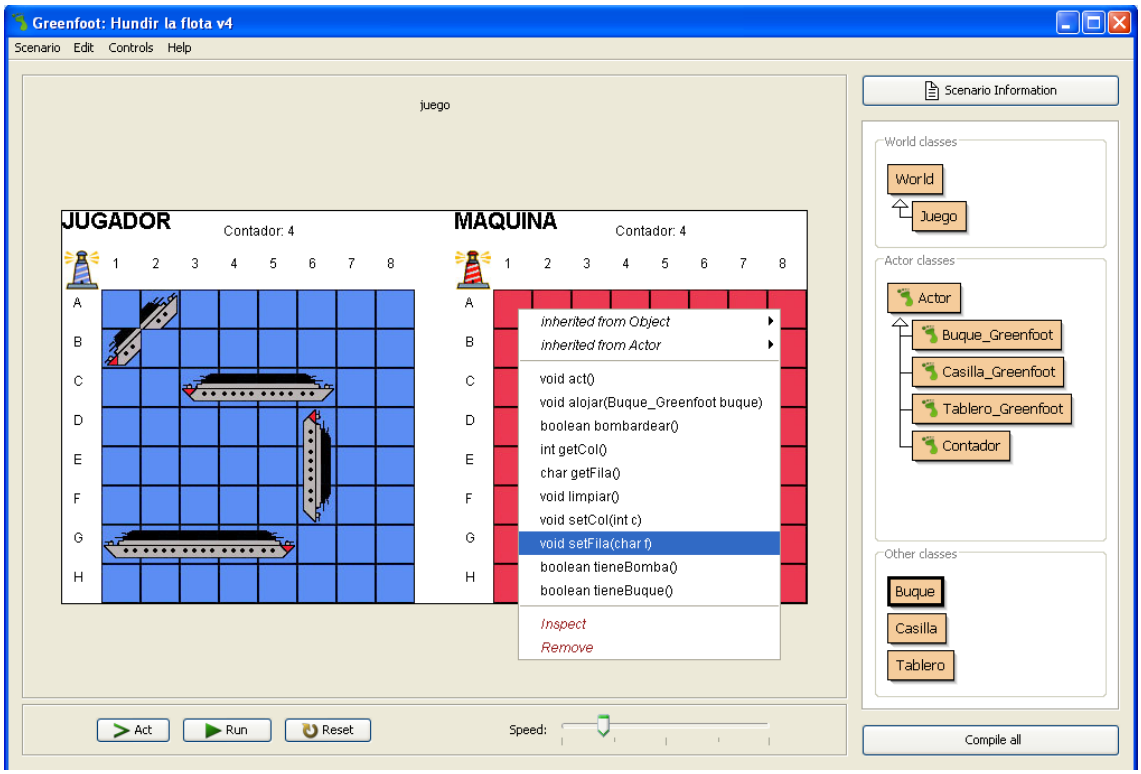
Como observamos en la Figura 29 el uso combinado de la herramienta, con el escenario proporcionado y las clases desarrolladas por los alumnos y añadidas a Greenfoot, los alumnos pueden distinguir de una manera intuitiva y visual la diferencia entre clase y objeto.

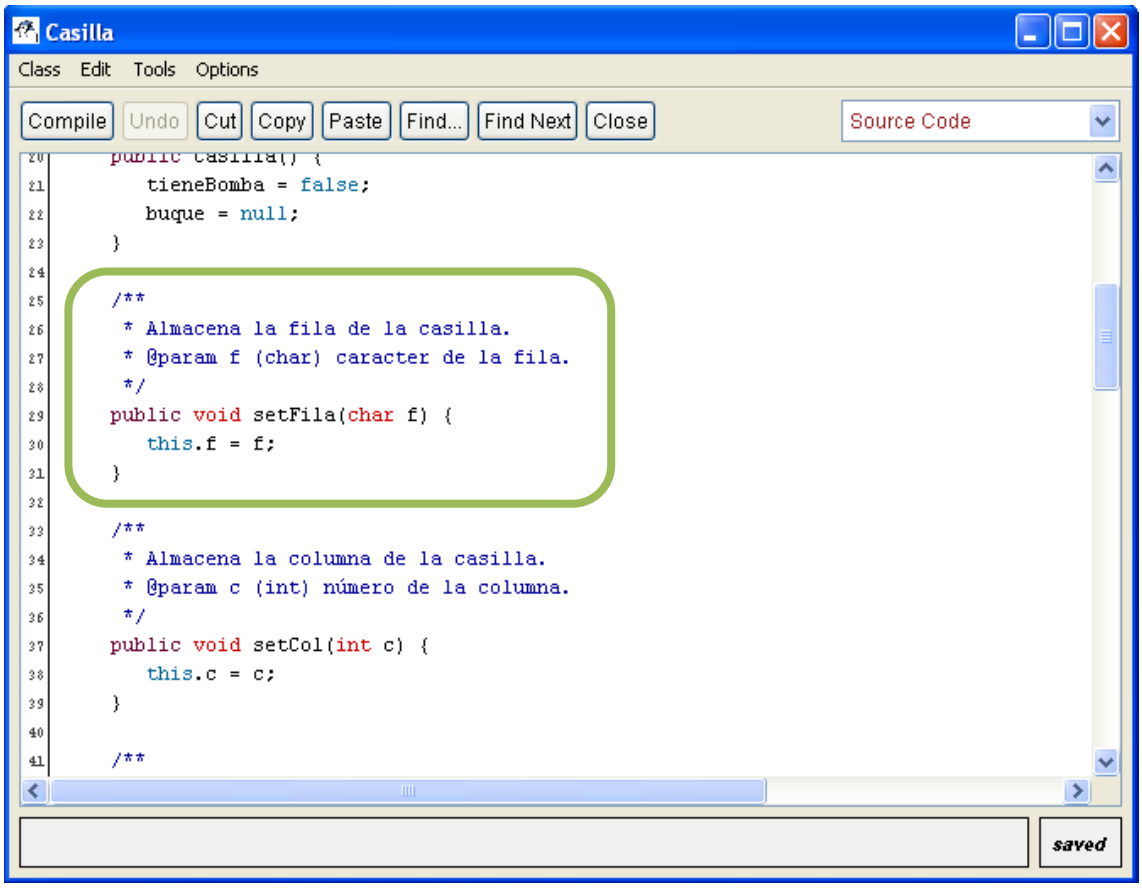
En el juego que aparece creado por defecto tras compilar todas las clases se pueden observar por ejemplo varios objetos buque los cuales mediante el inspector podemos ver que se trata de dos instancias de la clase Buque_Greenfoot. Dicha clase aparece en el menú de clases que proporciona Greenfoot a través del cual podemos acceder al código y crear un nuevo objeto buque, de manera que los alumnos vean dicha instanciación.

Invocación de métodos y propiedades de los objetos

La intención de estos objetivos tal y como se describió en la introducción son que los alumnos comprendan, cómo para indicar a un objeto que realice una tarea es necesario enviarle un mensaje y cómo para que un objeto procese el mensaje que recibe, la clase debe poseer un método que coincida con ese mensaje. Y puedan acceder al estado de los objetos con la intención de comprender su comportamiento.




Greenfoot: Hundir la flota v4
Scenario Edit Controls Help
juego
JUGADOR
Contador: 4
MAQUINA
Contador: 4
inherited from Object
inherited from Actor
void act()
void alojar(Buque_Greenfoot buque)
boolean bombardear()
int getCol()
char getFila()
void limpiar()
void setCol(int c)
void setFila(char f)
boolean tieneBomba()
boolean tieneBuque()
Inspect
Remove
Act
Run
Reset
Speed:
Scenario Information
World classes
World
Juego
Actor classes
Actor
Buque_Greenfoot
Casilla_Greenfoot
Tablero_Greenfoot
Contador
Other classes
Buque
Casilla
Tablero
Compile all


Casilla
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
public Casilla() {
tieneBomba = false;
buque = null;
}
/**
* Almacena la fila de la casilla.
* @param f (char) caracter de la fila.
*/
public void setFila(char f) {
this.f = f;
}
/**
* Almacena la columna de la casilla.
* @param c (int) número de la columna.
*/
public void setCol(int c) {
this.c = c;
}
/**
saved

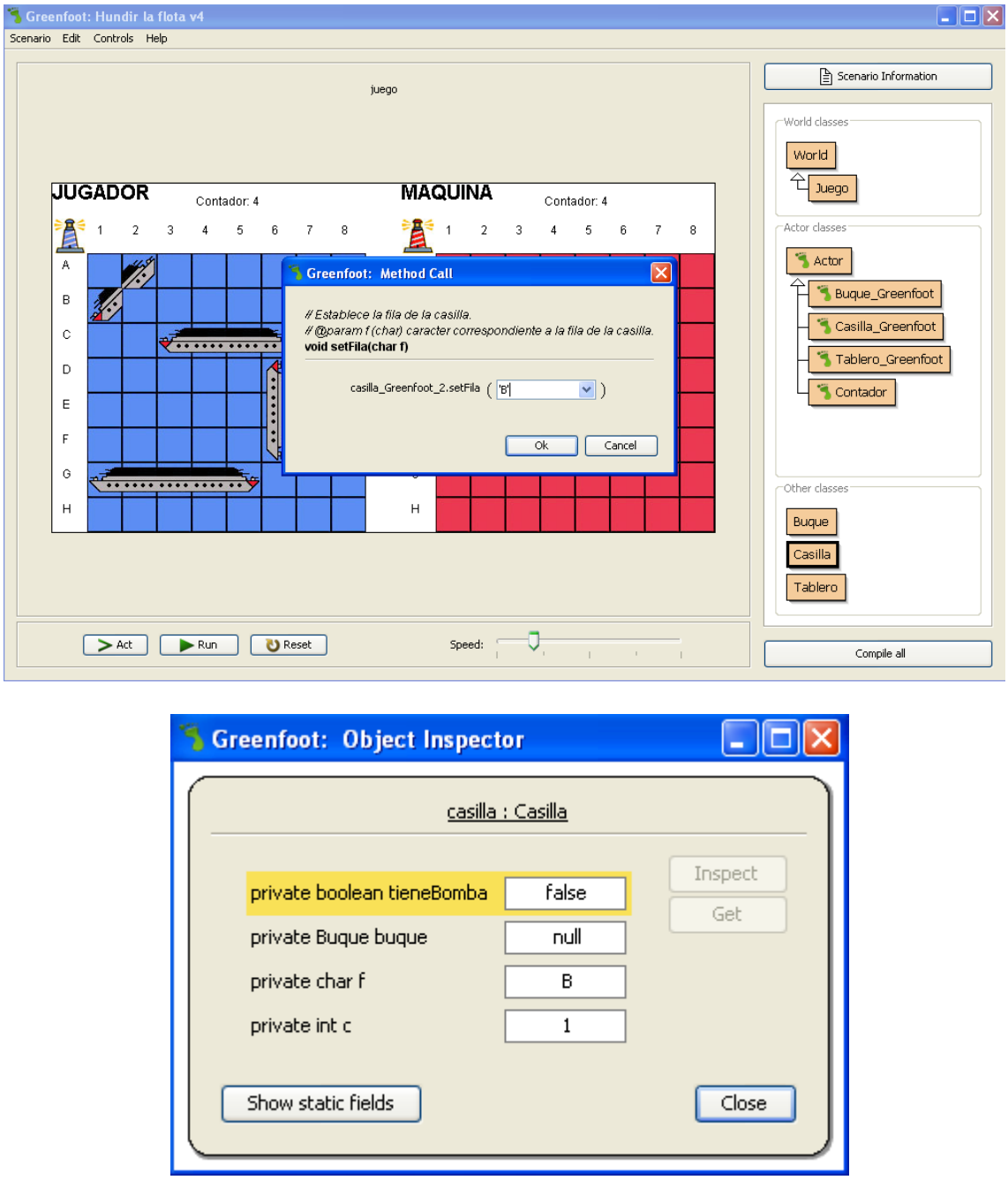


Figura 41. Secuencia de imágenes del objetivo: Invocación de métodos y propiedades de los objetos

En la secuencia de imágenes mostradas (Figura 41) se observa como el alumno puede acceder al estado del método inspeccionando el objeto, en este caso un objeto casilla perteneciente al tablero de la máquina. Como observamos se trata de la casilla A-1 la cual no tiene alojado ningún buque.

El alumno puede mediante la interacción con el objeto (haciendo click con el botón derecho sobre el objeto) invocar al método setFila(char f) que aparece en el menú desplegable con todos los métodos que contiene ese objeto. Y como se observa en la siguiente figura, ese método tiene su correspondiente código en la clase Casilla.

Una vez invocado el método se le pasan los parámetros correspondientes y se puede inspeccionar el objeto de nuevo viendo como la implementación del método ha hecho su funcionalidad y ha cambiado la fila de la casilla por el parámetro que se le había pasado.

Se pretende de esta manera demostrar la fácil comprensión de la invocación de métodos, como los alumnos pueden acceder al estado de los objetos, donde se muestra el identificador del objeto, de manera que ellos pueden ver en todo momento que se trata del mismo objeto y no de otro distinto, y además pueden ver como el estado ha cambiado debido a la invocación del método (debido a que la implementación del método lo quiere así).

Demostrados los sub-objetivos descritos se cumple así el objetivo principal que consistía en proporcionar un mecanismo visual que permitiera una representación de ciertos conceptos de la orientación objetos en Java mediante la simulación de un escenario.

En la segunda aportación destacada, la introducción de los usuarios en el uso de herramientas de apoyo, viene desencadenada del uso de la herramienta Greenfoot.

En muchas ocasiones los alumnos se conforman con los recursos que les ofrecen los profesores en las clases. Son pocos los que ponen en práctica ese espíritu investigador y de querer saber más que caracteriza a los estudiantes de ciencias y es por ello que quizá el uso de esta herramienta les invite a pensar en buscar una herramienta que consideren adecuada para cada momento. Pues tras el estudio realizado y presentado en el Apartado 2.2, sobre algunas de las herramientas existentes, se buscó y seleccionó la herramienta que se consideraba más adecuada y que cubriese de una manera más completa los objetivos planteados. Sin embargo esto no quiere decir que esta herramienta, se pueda considerar la herramienta por excelencia para la programación orientada a objetos. Cada herramienta aporta distintas funcionalidades que pueden ayudarnos a cubrir nuestros propios objetivos, y a medida que aumenta el nivel de conocimiento, estos objetivos pueden variar considerablemente.

Por ello se pretende abrir camino a los estudiantes en el uso de herramientas de apoyo que puedan facilitarles bien sea el aprendizaje de la programación orientada a objetos como cualquier otro aprendizaje que quieran llevar a cabo.

9.2. Trabajos futuros

Llegados a este punto final, se presentarán a continuación los trabajos futuros esperados, así como las mejoras que podrían realizarse en el presente proyecto.

- **Implantación de la solución proporcionada**: El principal trabajo futuro se refiere a la implantación de la solución en las clases de programación de Ingeniería de Telecomunicación. La realización y evaluación mediante el uso de cuestionarios que proporcionarán una idea de la utilidad de la solución proporcionada, permitiendo así mejorar todos aquellos aspectos que se consideren necesarios para facilitar la comprensión de la programación orientada a objetos en Java.

- **Posicionamiento de buques**: Este trabajo está centrado en la mejora del juego desarrollado. Consiste en permitir a los alumnos la colocación de los buques en el tablero arrastrando únicamente los buques. Los buques de cada tablero aparecerían creados y los alumnos deberán seleccionarlos y arrastrarlos hasta situarlos en el tablero en el lugar que deseen, pudiendo cambiar la dirección del buque mediante la pulsación de teclas.

- **Bombardeo de casillas**: Centrado también en el desarrollo del juego. Se le añadiría "inteligencia" al bombardeo de casillas, de manera que si tras el bombardeo de una casilla de manera aleatoria un buque fuese alcanzado, las bombas que serían lanzadas en los siguientes turnos, estarían alrededor de la casilla que ha sido tocada.

- **Motivación:** Se pretende que los alumnos se motiven con el uso de la herramienta seleccionada para profundizar en el mundo de la programación orientada a objetos en Java. Es por ello que sería necesario el desarrollo otras prácticas adaptadas para el uso de la herramienta Greenfoot.

9.3. Problemas encontrados

A lo largo del desarrollo del proyecto que se ha presentado como Proyecto Fin de Carrera se han encontrado distintas dificultades que han influido en numerosas ocasiones en el diseño inicial realizado sobre el juego así como en la implementación.

La primera dificultad encontrada ha sido derivada del la falta de conocimiento de la programación orientada a objetos. En Ingeniería Técnica en Informática de Gestión, dichos conceptos son impartidos de forma ligera y prácticamente irrelevante, lo que ha sido importante a la hora de realizar la implementación.

La segunda dificultad dada es el desconocimiento de la herramienta Greenfoot, a pesar de haber realizado un estudio sobre ella, leído numerosos tutoriales, utilizado escenarios proporcionados por la web, así como haber participado en un foro de discusión específico para la herramienta, un conocimiento profundo sobre ella sólo es adquirido a medida que se obtiene experiencia mediante el desarrollo de escenarios. Es totalmente imposible conocer aquellas cosas que condicionadas por la herramienta no pueden desarrollarse, hasta que no se ha dado tal caso.

Una vez iniciado el proceso de implementación, fue necesario cambiar el diseño inicial del diagrama de clases. A pesar de que la herramienta Greenfoot es muy útil y fácil de utilizar, el uso del API de Greenfoot ha condicionado el diseño. Se ha procurado que los alumnos de manera inicial tengan el mínimo conocimiento e interacción con código de Greenfoot, intentando de esta manera que obtengan los beneficios de la herramienta sin necesidad de aprender cómo se puede mediante Greenfoot añadir un objeto de manera visual.

Uno de los inconvenientes que tiene la herramienta, considerado de mínima importancia, es debido a que en ocasiones es necesario utilizar una llamada al método “addObject” (visible en el API de Greenfoot) para poder visualizar el objeto. El objeto es creado pero no añadido al mundo sin la llamada a este método. Es por esta y otras razones por las que se decidió cambiar el diseño.

Todas estas dificultades han supuesto un retraso, respecto a la planificación inicial del Proyecto Fin de Carrera, junto con el respectivo aumento de la carga de trabajo que se estimaba inicialmente.

9.4. Opiniones personales

En el Apartado 10.1 se comentaban aquellas aportaciones que se han realizado con el desarrollo del presente proyecto. Por ello considero necesario comentar en esta ocasión las aportaciones que me ha proporcionado el proyecto a mí, tanto desde el punto de vista técnico (en lo referente a conocimientos adquiridos) como desde el punto de vista personal.

El desarrollo del citado proyecto no solo tiene como objetivo proporcionar una solución a un problema planteado o la finalización de los estudios. Se pretendía que el proyecto llevado a cabo me proporcionase conocimientos de programación orientada a objetos y de sintaxis Java. Dicho objetivo considero ha sido cumplido, no sólo he aprendido programación orientada a objetos y sintaxis Java, he aprendido a llevar a cabo el desarrollo completo de un proyecto, y sobre todo y lo más importante a afrontar todas aquellas adversidades que pueden surgir en dicho desarrollo.

Tengo que destacar como positiva la experiencia, especialmente en la implementación de la aplicación, pues han sido los problemas surgidos en esta proceso los que más han ayudado a comprender la programación orientada a objetos y lo que más ha hecho que profundice en el mundo del lenguaje de programación Java.

Considero que el desconocimiento de la herramienta y el lenguaje, son los que me han ayudado en muchas ocasiones a comprender la dificultad planteada en el proyecto, esa dificultad de comprender los conceptos de programación orientada a objetos y cómo la herramienta Greenfoot junto con el escenario podría ayudar a comprenderlos.

Supongo que habrá gente que una vez terminado el proyecto se desentienda, o despreocupe, no sólo por cuestión de interés sino por cuestión de objetivos personales. Este proyecto ha despertado en mí un objetivo personal que me gustaría cumplir. Se trata de comprobar si realmente la solución planteada cumple en las aulas el objetivo buscado. Si la aplicación proporciona la utilidad para la que ha sido desarrollada y si esto llevaría a la implantación de la utilización de un nuevo mecanismo de enseñanza, así como futuros proyectos fin de carrera.

Comentar en último lugar y no por ello menos importante, la relación personal mantenida con la directora del proyecto. Desde mi punto de vista es importante tener un buen tutor en el proyecto, que sepa guiarte y ayudarte a lo largo del

mismo. Y en este sentido debo considerarme afortunada, pues la dedicación de la directora, la comprensión y la paciencia han sido primordiales para el desarrollo del proyecto.

10. Bibliografía

Referencias de artículos y libros

- Caitlin Kelleher, Motivating Programming: using storytelling to make computer programming attractive to middle school girls, November 2006.

- Dr. Robert B.K. Dewar, AdaCore Inc., Dr. Edmond Schonberg, AdaCore Inc., Where Are the Software Engineers of Tomorrow?.

- M. Ben-Ari, N. Ragonis, R. Ben-Bassat Levy, A Vision of Visualization in Teaching Object-Oriented Programming, Department of Science Teaching, Weizmann Institute of Science, Rehovot 76100 Israel.

- Michael Kölling, Poul Henriksen, Game Programming in Introductory Courses With Direct State Manipulation., Proceedings of ITiCSE'05, Lisbon, Portugal, June 2005.

- Poul Henriksen, A Direct Interaction Tool for Object-Oriented Programming Education, Master Thesis, The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, 2004.

- Poul Henriksen, Michael Kölling, in Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA), pages 73-82, Vancouver, BC, CANADA, November 2004.

- Stephen Cooper, Wanda Dann, Randy Pausch, Teaching Objects-first In Introductory Computer Science.

Referencias electrónicas

[1] BlueJ

http://www.bluej.org

[2] Jeliot3

http://cs.joensuu.fi/~jeliot

[3] Alice

http://www.alice.org/

[4] Greenfoot

http://www.greenfoot.org/

[5] COCOMO II (Descarga herramienta)

http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

[6] Manual de usuario de COCOMO II

http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/lab/cocomo/pgsi-p2-resumenmanual.pdf

[7] Transformación de líneas de código físicas a SLOC

http://www.ceh.nasa.gov/webhelpfiles/Cost_Estimating_Handbook_NASA_2004.htm#Software_Estimation.htm

[8] JGrasp

http://www.jgrasp.org/

Anexo. ¿Qué es Greenfoot?

1) Introducción a Greenfoot

Greenfoot es una herramienta software diseñada para proporcionar a los principiantes cierta experiencia en la programación orientada a objetos. Apoya el desarrollo de aplicaciones gráficas en el lenguaje de programación Java.

Esta herramienta fue diseñada e implementada por la Universidad de Kent (Inglaterra) y la Universidad Deakin (Melbourne, Australia).

El diseño de Greenfoot está inspirado originalmente considerando la combinación de características de dos de los más populares tipos de entornos de enseñanza: "microworlds" (micromundos), como "Karel the Robot" y entornos de interacción directa como "BlueJ".

Greenfoot aporta un sofisticado meta-framework interactivo que hace fácil la creación de variados micromundos, al mismo tiempo que proporciona la visualización del comportamiento e interacción directa entre objetos**.** [Poul Henriksen,2004]

a) Consideraciones de diseño.

En esta sección se presentan los argumentos que influyen en el diseño y desarrollo de Greenfoot.

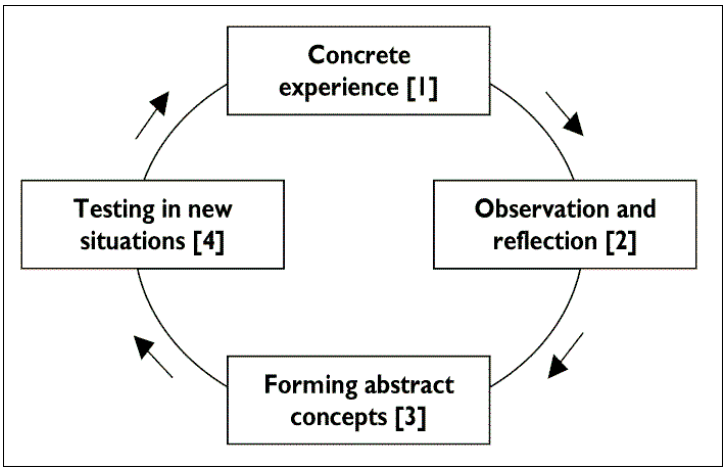
i) Círculo del aprendizaje a través de la experiencia de Kolb

El círculo de aprendizaje de Kolb presenta un modelo a veces usado para razonar sobre el proceso de aprendizaje, incluyendo el aprendizaje de los conceptos de programación. En la enseñanza de la programación temprana, especialmente en la enseñanza de orientación a objetos, puede ser difícil crear actividades en los cuatro cuadrantes del círculo.

La creación de experimentos activos y a partir de una experiencia concreta (que nos conduce a una observación reflexiva) en una aproximación a la orientación a objetos, se dificulta a menudo por obstáculos técnicos. Sintaxis oscura y problemas detallados en el entorno de programación a veces fuerzan a la experiencia concreta a estar al nivel de declaraciones en lugar de superiores abstracciones conceptuales.

La interpretación de un rol es una técnica que ha sido usada satisfactoriamente en el pasado para superar partes del problema en las primeras etapas de un curso. Cualquier herramienta debe proporcionar una aproximación a la orientación a objetos para principiantes que debe intentar apoyar las etapas prácticas del círculo de Kolb (experimentación, experiencia y observación) explícitamente al nivel del concepto fundamental: objetos.

En otras palabras, los estudiantes deben ser capaces de manipular, experimentar y observar objetos, no únicamente líneas de código fuente.



El círculo de aprendizaje de Kolb, con sus 4 etapas.

b) Crear interés

Un aspecto en el que la audiencia de destino en los niveles de enseñanza primaria difiere significativamente de los estudiantes universitarios o de secundaria, es en el nivel de compromiso con el objeto de estudio. Los estudiantes de niveles tempranos no han tomado una decisión consciente de participación en el estudio de la programación. Pueden frecuentemente no tener interés en la materia o incluso tener prejuicios contra ella. Así, el objetivo de una herramienta de apoyo para niveles tempranos de escolaridad no debe ser solo para ilustrar los importantes conceptos para los estudiantes, sino que debe generar interés en la materia en primer lugar.

Las actividades que los estudiantes van a realizar a través de la herramienta deben ser atractivas y relevantes para los estudiantes. Para estas dos características no existe una receta simple para cualquier estudiante, se encuentran algunas observaciones generales: un sistema que es interactivo, visual y permite la experimentación, genera curiosidad sin requerirse un primer estudio teórico y crea compromiso en el estudiante. Si un sistema es percibido como relevante para cualquier estudiante depende altamente de la educación del estudiante y también entra en el terreno de lo personal, no hay por tanto una única solución para cumplir estos objetivos.

Para el sistema de diseño de Greenfoot por tanto, es necesario esforzarse en que exista una flexibilidad en los escenarios para los estudiantes, que permita a los profesores dirigir a los estudiantes a nivel individual.

c) Apoyo a los docentes

Otro aspecto que distingue la situación de la enseñanza en los colegios de la de las universidades es el nivel de preparación que puede ser esperado de los profesores.

Los profesores dentro del campo de la informática en los colegios, tienen significativamente menos preparación en este campo, menos tiempo y apoyo para mantenerse al tanto de los últimos avances y menos tiempo de preparación del material de estudio.

Como resultado de esto, es beneficioso ver a los profesores como el segundo grupo destinatario de la herramienta Greenfoot. Aparte de apoyar a los estudiantes en su aprendizaje, Greenfoot debe ser diseñado para apoyar a los profesores al mismo tiempo.

Esto se puede realizar de varias maneras: se puede organizar la presentación de conceptos importantes en la herramienta de manera que se anime a la temprana discusión de éstos.

Se puede diseñar la herramienta de manera que el compartir escenarios y ejercicios sea una tarea fácil. Existe por lo tanto una clara tensión entre la flexibilidad y el apoyo al profesor. Este apoyo al personal docente tiende a significar la provisión de una rígida estructura en la herramienta de apoyo, así que los profesores tienen menos trabajo y mejor guía en sus actividades de enseñanza. Permitir flexibilidad puede contradecir directamente esto. Uno de los desafíos del diseño de Greenfoot es encontrar una solución que permita que estos dos objetivos coexistan.

d) Objetivos

Tras describir los argumentos que conducen al desarrollo de esta herramienta, se describen ahora los objetivos.

El objetivo principal podría resumirse como “adecuado para la enseñanza de la orientación a objetos a nivel escolar”. Se debe apuntar que el centrarse en los primeros niveles no excluye el uso de la herramienta a nivel universitario, dado que en la actualidad no se re.

i) Experimentación y retroalimentación visual (visual feedback)

Se intenta que el sistema sea altamente visual e interactivo. Los usuarios deben ser capaces de experimentar con instanciaciones de conceptos directamente, vía interfaz de usuario y adquirir un entendimiento de conceptos importantes a través de la retroalimentación visual.

Se espera que con esto se cierre el círculo de actividades del modelo de Kolb discutido anteriormente y que contribuya satisfactoriamente al reto de atraer a estudiantes sin ningún interés importante por la programación.

Sorprendentemente un balance entre simplicidad y riqueza de la herramienta es importante. Si el programa no es simple en su uso, los usuarios podrían perder el interés y el uso de esta herramienta podría no ser productivo.

ii) Flexibilidad en los escenarios

Para atraer el interés de los estudiantes, el sistema necesita ser capaz de poseer ejemplos adecuados a su edad, género y características personales y culturales, además de otros factores individuales. Para el sistema de diseño, esto significa que Greenfoot, debe proveer de una gran variedad de diferentes actividades y escenarios. Creando flexibilidad en escenarios permite variar la complejidad y así el material va siendo aprendido según su nivel de dificultad.

iii) Clara enseñanza de conceptos de la orientación a objetos

El primer foco debe ser el desarrollo del entendimiento en los estudiantes de los conceptos usados en la programación orientada a objetos. Con el uso de la herramienta Greenfoot los estudiantes deben familiarizarse con los conceptos fundamentales de la orientación a objetos como las clases, objetos, invocación de métodos y conceptos imperativos de programación.

iv) Fácil desarrollo de escenarios y ejercicios

Parte del gran objetivo de proveer de un buen soporte para profesores es que éste debe aportar un fácil desarrollo de escenarios y ejercicios. Greenfoot debe intentar realizar el desarrollo de un ejercicio y de un

escenario lo suficientemente fácil para que muchos profesores puedan desarrollar sus propias versiones. Esto significa que el escenario a nivel de usuario debe estar separado de la estructura general de implementación.

v) Apoyo para la migración a otros entornos

El sistema Greenfoot debe ser diseñado para que los conceptos y conocimientos aprendidos puedan ser transferidos fácilmente a otros entornos como BlueJ que puede ser usado como el siguiente entorno de desarrollo.

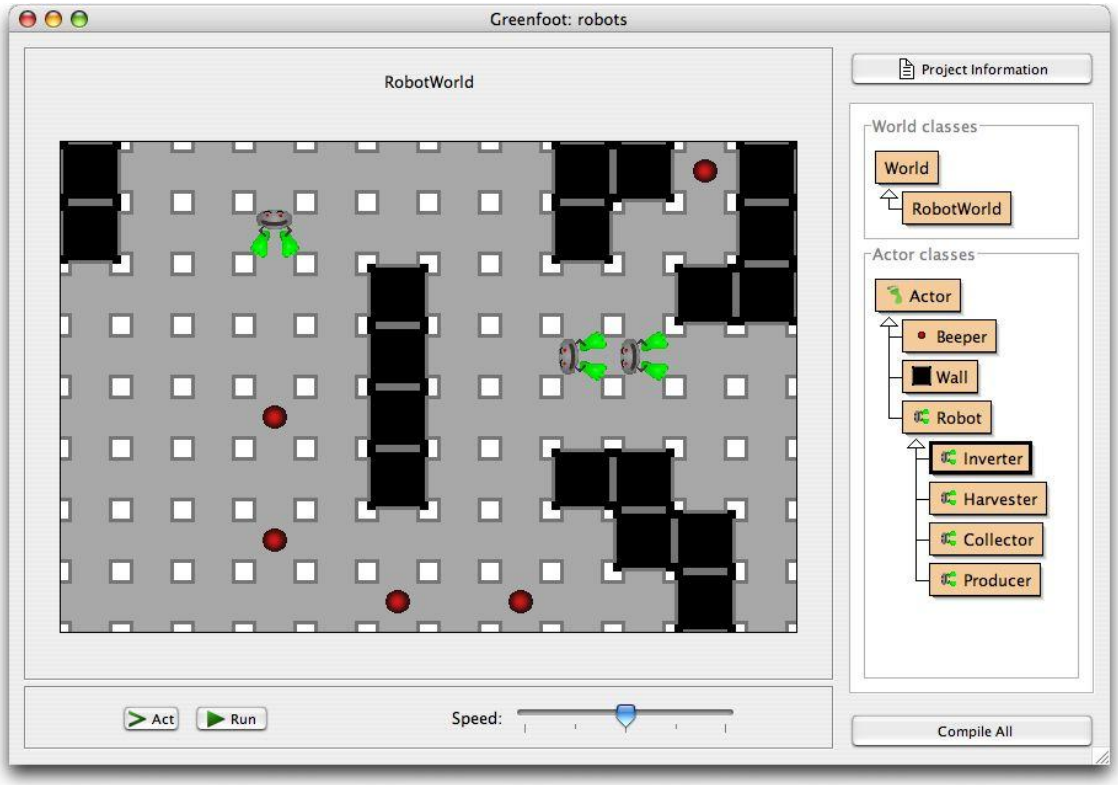
e) La herramienta Greenfoot

El sistema Greenfoot es una combinación de framework y entorno para la creación de aplicaciones de simulación interactivas en un plano bidimensional en el lenguaje de programación Java. Es adecuado para nuevos programadores.

En Greenfoot la visualización de objetos y la interacción entre ellos son los elementos clave.

Una manera de visualización del sistema es como una extensión del banco de objetos de BlueJ. Greenfoot extiende la idea del banco de objetos al mundo de los objetos. En este mundo los objetos tienen una apariencia gráfica y una posición en el mundo. La interacción directa con estos objetos es todavía posible, como en el original BlueJ pero el comportamiento de los objetos puede ser ahora observado directamente al ver los cambios en la posición y apariencia individual de los objetos.

El propio mundo de los objetos (visible como área subyacente detrás de los objetos Greenfoot) llega a ser un objeto programable, interactivo integrado en la estructura de la aplicación.



Mundo del escenario Robots

i) El sistema de interfaz visual

La mayor parte de la interfaz de usuario de Greenfoot está reservada para el expositor del mundo Greenfoot, mostrado en el centro de la pantalla. Éste dispone de los objetos Greenfoot.

A la derecha del mundo encontramos el menú de clases. Todas las clases que participan en la aplicación actual son mostradas con botones para compilación y creación de nuevas clases. Las clases son divididas en Greenfoot-World Classes y en Greenfoot-Object Classes, que serán explicadas más adelante.

La parte baja de la ventana posee el control de ejecución para ejecutar, parar o simular un solo paso de ejecución y un deslizador para controlar la velocidad de ejecución.

ii) Estructura del Escenario Greenfoot

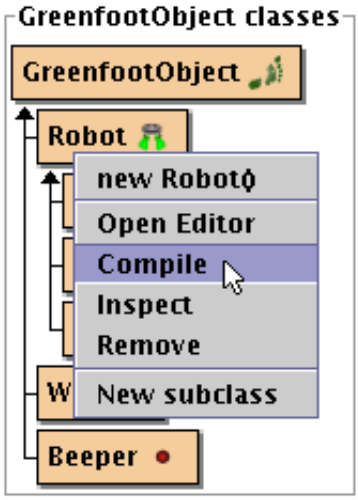
Todas las clases cuyas instancias deben ser visibles en el mundo Greenfoot amplían la superclase predefinida Greenfoot-Actor. El entorno también provee de una clase predefinida Greenfoot-World, que implementa el mundo mismo.

El mundo proporciona una red de celdas, que contienen los objetos Greenfoot. Cada objeto Greenfoot puede especificar su propia apariencia usando un icono o un método de dibujo. Los objetos Greenfoot tienen localización en el mundo y una rotación que es aplicada al icono.

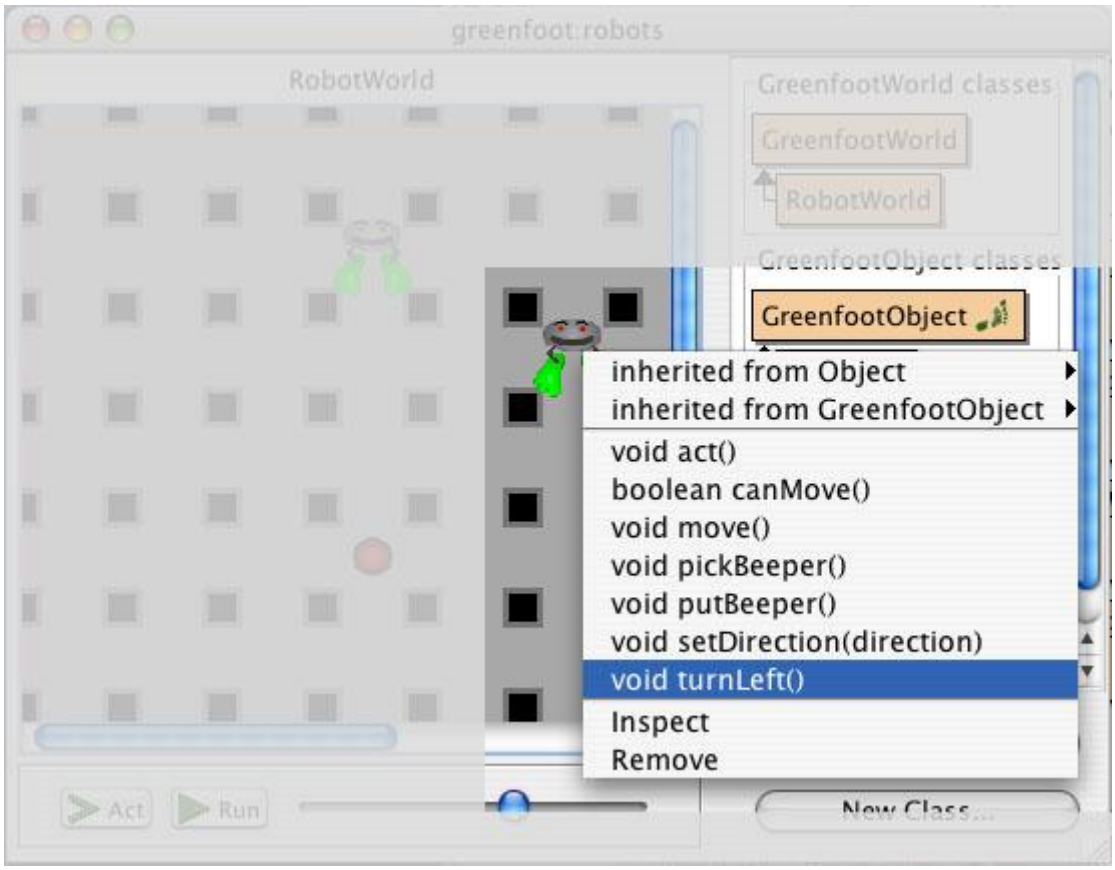
El mundo provee métodos, entre otros, para cambiar la resolución (esencialmente ajustando el tamaño de cada celda en pixels), para cambiar el tamaño del mundo (número de celdas), para colocar un fondo de imagen y para dibujar sobre su fondo. Usando estos métodos, los mundos que difieren enormemente en su apariencia pueden idearse como parte de la creación de un escenario.

Todos los objetos en el mundo Greenfoot son automáticamente animados e interactivos. Pueden tener un comportamiento que es exhibido cuando la simulación se ejecuta usando el botón “Run” y pueden ser usados para una interacción directa asociados entre menús “pop-up” cuando la simulación se encuentra en pausa.

La simulación de animación y las características de la interacción directa de objetos son construidas en el entorno Greenfoot. Un menú pop-up de objetos que contiene la lista de métodos que pueden ser invocados sobre el objeto así como una opción para la inspección total del estado del objeto.



Menú pop-up de una clase Greenfoot.



Menú pop-up de un objeto

iii) Greenfoot IDE (Integrated Development Enviroment-Entorno de desarrollo integrado)

El sistema Greenfoot es un entorno integrado: aparte de la interfaz principal descrita anteriormente, contiene un editor, un compilador y un depurador.

Es un sistema auto-contenido que proporciona todas las herramientas necesarias para el desarrollo, examen y ejecución de una aplicación completa. El tiempo de ejecución subyacente y el compilador usan el estándar de Java. Las clases Greenfoot son clases Java estándar. La implementación Greenfoot está basada en el sistema BlueJ y muchas de las herramientas BlueJ (editor, depurador, generador Javadoc...) están disponibles en Greenfoot con una forma muy similar a BlueJ.

- **Navegador de clases:** el entorno también aporta una vista de las clases que participan en la simulación en la parte derecha de la ventana principal. Estas clases pueden ser editadas, compiladas e instanciadas. A estas acciones se puede acceder desde el menú pop-up de clases.

  La creación de nuevas clases puede realizarse mediante la selección de “New subclass” de el menú pop-up de clases o clickeando el botón “New class” por debajo de el icono de la clase.

  El navegador de clases está dividido en dos secciones: **clases Greenfoot-Actor** son las clases que van a ser visualizadas en el mundo. Su superclase Greenfoot-Actor, siempre será mostrada en el navegador de clases. La clase Greenfoot-Actor no puede ser modificada.

  Las subclases de Greenfoot-Actor tendrán típicamente un icono individual. Este icono es mostrado en la representación de la clase al lado del nombre de la clase. Los objetos Greenfoot para los cuales no se especifica una apariencia tienen una vista por defecto definida en su superclase.

  Las **clases Greenfoot-World** son clases que representan mundos. Pueden existir diferentes mundos en un proyecto individual. La

superclase Greenfoot-World será siempre mostrada en el navegador de clases.

Las clases Greenfoot-World tienen menú pop-up exactamente como el de las ya descritas clases Greenfoot-Actor. Cuando un constructor es seleccionado por una subclase Greenfoot-World, el nuevo objeto automáticamente reemplaza el mundo existente en la vista principal de la interfaz de usuario Greenfoot.

f) Nuevas posibilidades de Greenfoot.

El uso de Greenfoot puede abrir nuevas posibilidades para el diseño que actualmente no se dan. Una de estas posibilidades es un efecto directo del desacoplamiento de los escenarios de la estructura: el uso de múltiples mundos en un solo proyecto.

Actualmente, cada micromundo conlleva unas consideraciones generales en su uso. Cada uno tiene diferentes requisitos técnicos, diferente procedimiento de instalación, diferentes interfaces de usuario que necesitan ser aprendidas y diferentes modelos de interacción. Como resultado, no hay suficiente tiempo en un solo proyecto para el uso de más de un escenario.

Desde que Greenfoot tiene una interfaz consistente y un modelo de interacción a través de los diferentes escenarios del mundo, las consideraciones de aprendizaje del uso de Greenfoot existen una única vez y el uso adicional de mundos es casi libre. Esto abre las posibilidades de usar una secuencia de escenarios simulados tal vez de mayor complejidad o de diferentes áreas de aplicación) en el mismo proyecto. Un proyecto podría comenzar con tortugas, seguir con robots y terminar con la implementación de una simulación de un ascensor para un rascacielos.