



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

**Desarrollo de una Aplicación Educativa para
Iniciarse en la Programación**

Realizado por: Octavio Martínez García

Dirigido por: Anselmo Peñas

Curso: 2013-14



Desarrollo de una Aplicación Educativa para Iniciarse en la Programación

Proyecto de Fin de Grado de modalidad *oferta general*

Realizado por: Octavio Martínez García

Dirigido por: Anselmo Peñas

Fecha de lectura y defensa: 23 / 09 / 2014

Índice general

1. Introducción	1
1.1. Descripción	1
1.2. Motivación	1
1.3. Posicionamiento en el mercado	2
1.4. Personal Involucrado	4
1.4.1. Descripción y objetivos de alto nivel	4
1.4.2. Entorno de Usuario	5
1.4.3. Objetivos a Nivel de Usuario	5
1.5. Visión General del Producto	5
1.5.1. Perspectiva del Producto	5
1.5.2. Resumen de las características del Sistema	5
1.5.3. Otros Requisitos y Restricciones	6
2. Requisitos	7
2.1. Modelo de Casos de Uso	7
2.1.1. Introducción	7
2.1.2. Lista Evento-Actor-Objetivo	7
2.1.3. Modelo de Casos de Uso	8
2.1.3.1. UC0 - Iniciar Entorno	8
2.1.3.2. UC1 - Colocar comando	10
2.1.3.3. UC2 - Unir dos comandos	12
2.1.3.4. UC3 - Mover comando	17
2.1.3.5. UC4 - Registrar evento sensor	21
2.1.3.6. UC5 - Ejecutar programa	23
2.1.3.7. UC6 - Parar programa	25
2.1.3.8. UC7 - Resetear programa	27
2.1.3.9. UC8 - Cargar fondo de escenario	28
2.1.3.10. UC9 - Cargar robot	30
2.1.3.11. UC10 - Gestionar proyectos	32
2.1.3.12. UC11 - Hacer login	32
2.1.3.13. UC12 - Limpiar Escritorio	32
2.2. Contratos de las operaciones	33
2.2.1. Introducción	33
2.2.2. Contratos	34
2.2.2.1. CO1: Iniciar Entorno	34
2.2.2.2. CO2: Arrastrar Comando	35

2.2.2.3. CO3: Acoplar Comando	35
2.2.2.4. CO4: Eliminar Comando	36
2.2.2.5. CO5: Colocar Comando	37
2.2.2.6. CO6: Elegir Comando	38
2.2.2.7. CO7: Mover Comando	39
2.3. Especificación Complementaria	40
2.3.1. Introducción	40
2.3.2. Funcionalidades	40
2.3.3. Usabilidad	40
2.3.4. Fiabilidad	41
2.3.5. Rendimiento	41
2.3.6. Soporte	42
2.3.6.1. Implementación	42
2.3.6.2. Instalación	42
2.3.6.3. Escalabilidad	42
2.3.6.4. Configurabilidad	42
2.3.6.5. Compatibilidad	43
2.3.7. Interfaces	43
2.3.7.1. Interfaces de Usuario	43
2.3.7.2. Interfaces con Sistemas Externos	45
2.3.8. Aspectos Legales	45
2.4. Glosario	45
2.4.1. Introducción	45
3. Diseño	47
3.1. Lenguaje de Comandos	47
3.1.1. Léxico (Tokens)	47
3.1.2. Sintaxis (Gramática)	48
3.1.3. Semántica (Acciones)	49
3.2. Arquitectura SW	50
3.3. Diagramas de Clases	51
3.3.1. DCD Ghost	51
3.3.2. DCD GUI	52
3.3.3. DCD Lenguaje	53
3.4. Diagramas de Interacción	54
4. Implementación	59
4.1. Tecnologías Utilizadas	59
4.2. Aspectos Especiales de Implementación	60
4.2.1. Lógica del Drag and Drop	60
4.2.1.1. DI GhostDnD()	61
4.2.2. Lógica de la Creación de Programas	61
4.2.3. Lógica de la Ejecución de Programas	63
4.2.3.1. Start	63

4.2.3.2. Stop	64
4.2.4. Semántica de la sentencia Wait	64
4.3. Arquitectura - Paquetes	65
4.3.1. Dominio	65
4.3.2. Ghost	65
4.3.3. GUI	65
4.3.4. Lenguaje	65
4.3.5. Utils	66
5. Pruebas	67
5.1. Comandos	67
5.1.1. Snap	67
5.1.1.1. Start:	67
5.1.1.2. Repeat:	67
5.1.1.3. Wait:	68
5.1.1.4. If:	68
5.1.1.5. End_Repeat:	69
5.1.1.6. End_If:	69
5.1.1.7. Move:	70
5.1.1.8. Set Position:	70
5.1.1.9. Set Speed:	70
5.1.1.10. Color:	71
5.1.1.11. Key:	71
5.1.1.12. GENERAL:	72
5.1.2. JComboBox / JFormattedTextField	72
5.1.2.1. Comando Move	72
5.1.2.2. Comando Set Position	73
5.1.2.3. Comando Set Speed	73
5.1.2.4. Comando Sensor Color	74
5.1.2.5. Comando Sensor Key	74
5.1.2.6. GENERAL	74
5.2. Creación de Programas	74
5.2.1. Start:	75
5.2.2. Repeat:	75
5.2.3. Wait:	76
5.2.4. If:	77
5.2.5. End_Repeat:	78
5.2.6. End_If:	78
5.2.7. Move:	79
5.2.8. Set Position:	79
5.2.9. Set Speed:	80
5.2.10. Color:	80
5.2.11. Key:	80

5.3. Ejecución de Programas	81
5.3.1. Test 1	81
5.3.2. Test 2	81
5.3.3. Test 3	81
5.3.4. Test 4	81
5.3.5. Test 5	81
5.3.6. Test 6	82
5.3.7. Test 7	82
5.3.8. Test 8	83
5.3.9. Test 9	83
5.3.10. Test 10	84
5.3.11. Test 11	84
5.4. Panel de Control	85
5.4.1. Botón Load Robot	85
5.4.2. Botón Load Background	85
5.4.3. Botón Start	85
5.4.4. Botón Stop	86
5.4.5. Botón Reset	86
5.4.6. Botón Clean	86
5.4.7. Sensor Ultima Tecla	86
5.4.8. Sensor Color Actual Robot	87
5.4.9. Sensor Coordenadas del Ratón	87
5.4.10. Sensor Color del Ratón	87
5.5. Mapas Cerrados	87
5.6. Mapas Abiertos	88
6. Despliegue	89
7. Gestión del Proyecto	90
7.1. Plan de Proyecto	90
7.1.1. Introducción	90
7.1.2. Organización del Proyecto	90
7.1.3. Prácticas y medidas del Proyecto	91
7.1.4. Calendario General del Proyecto	93
7.2. Planes de Iteración	95
7.2.1. Plan de Iteración T2	95
7.2.1.1. Iteración Actual	95
7.2.1.2. Objetivos de Alto Nivel	95
7.2.1.3. Items de Trabajo	96
7.2.1.4. Problemas y Aspectos Abiertos	96
7.2.1.5. Valoraciones	96
7.2.2. Plan de Iteración T1	97
7.2.2.1. Iteración Actual	97
7.2.2.2. Objetivos de Alto Nivel	97

7.2.2.3.	Items de Trabajo	98
7.2.2.4.	Problemas y Aspectos Abiertos	98
7.2.2.5.	Valoraciones	99
7.2.3.	Plan de Iteración C5	100
7.2.3.1.	Iteración Actual	100
7.2.3.2.	Objetivos de Alto Nivel	100
7.2.3.3.	Items de Trabajo	101
7.2.3.4.	Problemas y Aspectos Abiertos	101
7.2.3.5.	Valoraciones	102
7.2.4.	Plan de Iteración C4	103
7.2.4.1.	Iteración Actual	103
7.2.4.2.	Objetivos de Alto Nivel	104
7.2.4.3.	Items de Trabajo	104
7.2.4.4.	Problemas y Aspectos Abiertos	105
7.2.4.5.	Valoraciones	106
7.2.5.	Plan de Iteración C3	107
7.2.5.1.	Iteración Actual	107
7.2.5.2.	Objetivos de Alto Nivel	108
7.2.5.3.	Items de Trabajo	108
7.2.5.4.	Problemas y Aspectos Abiertos	109
7.2.5.5.	Valoraciones	109
7.2.6.	Plan de Iteración C2	110
7.2.6.1.	Iteración Actual	110
7.2.6.2.	Objetivos de Alto Nivel	111
7.2.6.3.	Items de Trabajo	112
7.2.6.4.	Problemas y Aspectos Abiertos	113
7.2.6.5.	Valoraciones	113
7.2.7.	Plan de Iteración C1	115
7.2.7.1.	Iteración Actual	115
7.2.7.2.	Objetivos de Alto Nivel	115
7.2.7.3.	Items de Trabajo	116
7.2.7.4.	Problemas	117
7.2.7.5.	Valoraciones	118
7.2.8.	Plan de Iteración E3	118
7.2.8.1.	Iteración Actual	118
7.2.8.2.	Objetivos de Alto Nivel	118
7.2.8.3.	Items de Trabajo	119
7.2.8.4.	Problemas	119
7.2.8.5.	Valoraciones	120
7.2.9.	Plan de Iteración E2	121
7.2.9.1.	Iteración Actual	121
7.2.9.2.	Objetivos de Alto Nivel	121
7.2.9.3.	Items de Trabajo	122
7.2.9.4.	Problemas	122

7.2.9.5. Valoraciones	123
7.2.10. Plan de Iteración E1	123
7.2.10.1. Iteración Actual	123
7.2.10.2. Objetivos de Alto Nivel	124
7.2.10.3. Items de Trabajo	124
7.2.10.4. Problemas	125
7.2.10.5. Valoraciones	125
8. Extensiones	126
9. Bibliografía	127
10. Anexo A: Manual de Usuario	128
10.1. Instalación	128
10.2. Interfaz de Usuario	128
10.2.1. Escenario:	128
10.2.2. Panel de Control:	129
10.2.2.1. Botones:	130
10.2.2.2. Indicadores	133
10.2.3. Paleta de Comandos:	134
10.2.4. Escritorio	139
10.3. Mapas	140
10.3.1. Mapas cerrados	140
10.3.2. Mapas abiertos	142
10.4. Créditos	144
11. Anexo B: Lista de Figuras	146
12. Anexo C: Palabras Clave para Búsquedas	149

1 Introducción

1.1. Descripción

El proyecto se trata de desarrollar una aplicación educativa que permita a alumnos de primaria familiarizarse con la programación y desarrollar sus primeros programas.

Se trata de un entorno gráfico, donde los alumnos podrán utilizar sentencias del lenguaje de programación gráfico diseñado, consistentes en imágenes cargadas en la aplicación, para construir programas que controlen las acciones de un robot, representado por otra imagen o pequeña animación, en un escenario, el cual vendrá dado por una imagen de fondo cargada desde un archivo.

1.2. Motivación

Considero que el conocimiento de la programación es una valiosa herramienta en un mundo cada vez mas vinculado al fenómeno digital, donde las personas están constantemente expuestas a todo tipo de dispositivos, y rodeadas de las mas diversas aplicaciones informáticas en prácticamente todos los ámbitos de la vida.

De este modo la posibilidad de enseñar a los jóvenes a pensar en términos de un lenguaje de programación es una gran oportunidad que se debería aprovechar, ya que al margen de la propia habilidad para codificar, se adquieren otras valiosas habilidades derivadas, como pueden ser la capacidad para resolver problemas, pensamiento abstracto, facilidad para comunicar ideas, creatividad, pensamiento lógico-analítico, etc., habilidades todas ellas que les serán de utilidad en cualquier ámbito de trabajo futuro.

Sin embargo, el aprendizaje de un lenguaje de programación puede resultar una tarea demasiado abstracta y compleja, máxime si tenemos en cuenta que los alumnos van a ser niños de entre 6 y 12 años, de modo que es necesario hacer el aprendizaje mas accesible, pasando a un estilo visual-kinestésico, donde el alumno pueda identificar un problema concreto, y buscar y aplicar las herramientas a su disposición para resolver dicho problema, aprendiendo de este modo bajo un enfoque descendente a usar dichas herramientas.

Bajo este enfoque se pretende incentivar la curiosidad de los alumnos, haciendo que se interesen por un problema, y que sean ellos mismos los que busquen formas de llegar a resolverlo, proceso mediante el cual se adquirirá el conocimiento de una forma aplicada.

Este es el método de aprendizaje que se pretende aplicar al usar la aplicación que se va a desarrollar, de modo que los alumnos puedan identificar un problema que se presentaría como una escena gráfica, compuesta de un fondo y un robot que se movería sobre ese fondo, y una serie de comandos que formarían el lenguaje de programación gráfico que

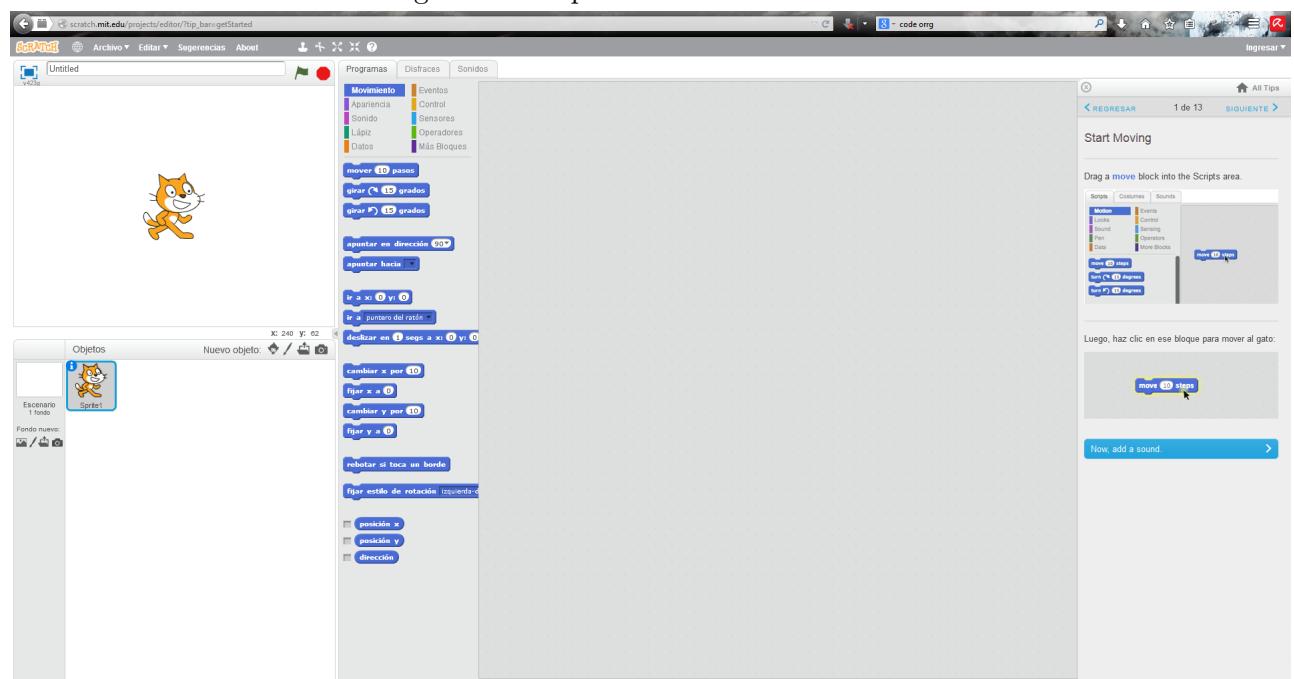
el alumno puede utilizar para resolver el problema, es decir, para que el robot se mueva de una determinada manera sobre el escenario.

1.3. Posicionamiento en el mercado

Actualmente existen otras soluciones encaminadas a resolver el mismo problema o a proporcionar una funcionalidad similar a la del sistema que se pretende construir, entre las mas destacadas podemos citar:

- Scratch:

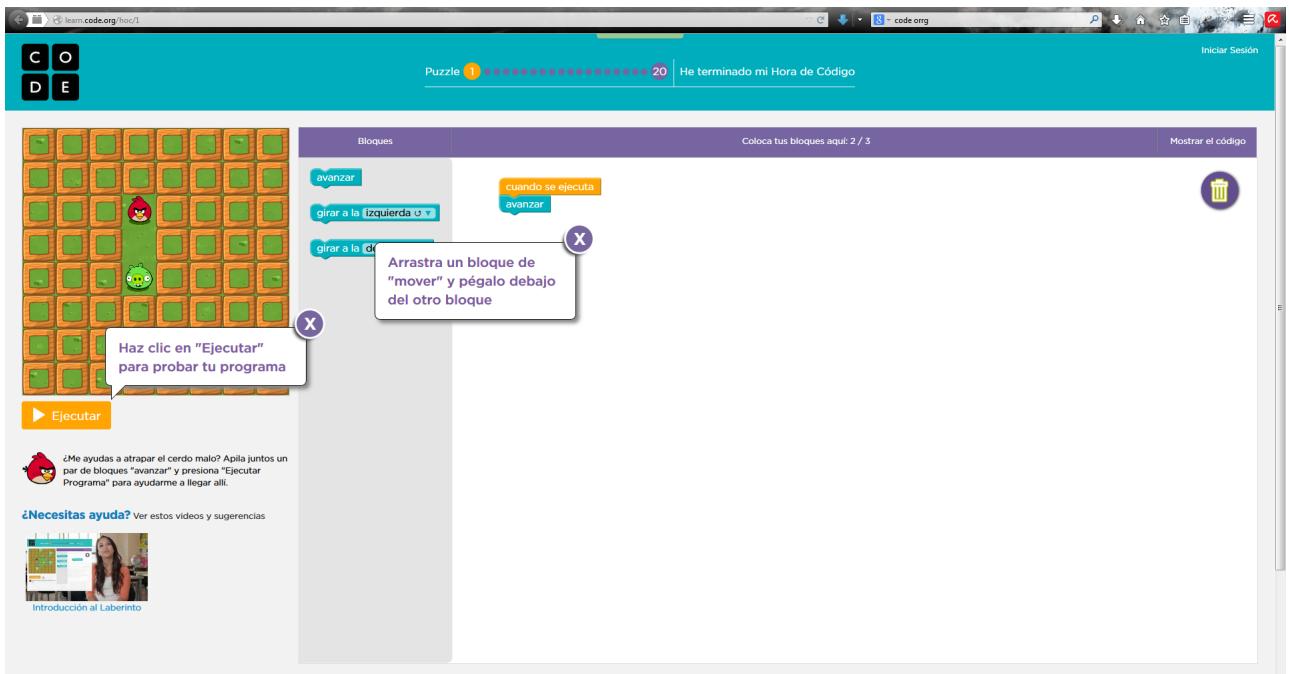
Figura 1.1: Captura Scratch



- Se trata de un entorno de programación y una comunidad donde los estudiantes pueden crear y compartir sus propios programas mediante un lenguaje de programación visual compuesto por bloques
- El punto fuerte de Scratch es que enseña a los estudiantes a pensar creativamente, razonar sistemáticamente, y trabajar colaborativamente.
- Sin embargo, en Scratch empiezas con un «lienzo en blanco» para crear un programa, lo cual incentiva la creatividad, pero se echan en falta una serie de ejercicios que hagan de «guia», que faciliten al alumno el familiarizarse con el lenguaje a través de la resolución de una serie de problemas concretos

- Code.org

Figura 1.2: Captura Code.org



- Es un proyecto que pretende acercar la programación a personas sin base en este ámbito, mediante la realización de diversos ejercicios guiados que requieren la utilización de las diferentes sentencias del lenguaje de programación que utiliza (similar al de Scratch)
- En este caso el «punto débil» del sistema es que carece de ese aspecto de libertad creativa que proporciona Scratch

En principio el sistema a desarrollar es de una dimensionalidad menor que los anteriores sistemas, y está enfocado a un público mas limitado (niños de educación primaria), si bien se contempla la posibilidad de que se pudiera extender su funcionalidad o su público objetivo en un futuro, por lo cual para su desarrollo debería facilitarse su modularidad y escalabilidad.

El sistema intenta aunar los dos tipos de funcionalidades de los sistemas mencionados arriba, usando 2 tipos de modos de uso, instrumentados a través de los 2 tipos de mapas:

- Mapas Abiertos: mapas libres para permitir experimentar con el comportamiento del Robot y crear programas libremente
- Mapas Cerrados: para guiar a los alumnos hacia el aprendizaje de las diversas sentencias de programación, a través de varios niveles de mapas con un camino definido y otros elementos opcionales con que interactuar

Por su parte el sistema no precisa de conexión a internet para su uso, ya que se trata de un ejecutable y un directorio de imágenes incluidos en el paquete de instalación, teniendo la ventaja añadida de que se trata de una aplicación ligera y muy fácil de instalar, que si bien inicialmente esta pensada como una aplicación de escritorio, sería posible portarla para su uso en dispositivos móviles.

1.4. Personal Involucrado

1.4.1. Descripción y objetivos de alto nivel

A continuación se detalla una posible lista del personal que tendría intereses en el proyecto, y cuales serían sus posibles objetivos en relación con la aplicación. Esta lista es la que se ha utilizado inicialmente en el seno del desarrollo iterativo, como base para el análisis de requisitos del proyecto, dentro del Modelo de Casos de Uso.

- Alumnos: estudiantes de educación primaria
 - crear / inventar
 - divertirse
 - aprender
 - relacionarse
- Profesores: docentes de educación primaria
 - promover el aprendizaje de sus alumnos
 - incentivar la curiosidad del alumnado
 - guiar a sus alumnos
- UNED: comunidad de profesores y alumnos de la UNED
 - evaluar la adecuación y desarrollo del sistema
 - promover la distribución del sistema si su calidad es la esperada
- Jóvenes Inventores: asociación educativa que trata de promover la creatividad y curiosidad científica en niños de primaria
 - promover el deseo de crear
 - incentivar la curiosidad de los alumnos
 - poner a disposición de los alumnos los medios y conocimientos técnicos necesarios que les permitan crear
- Desarrollador:
 - verificar el correcto funcionamiento del sistema
 - crear y adaptar el sistema de acuerdo a las necesidades del personal involucrado
 - ampliar el sistema con posibles nuevas funcionalidades

1.4.2. Entorno de Usuario

El Sistema en principio está pensado para ser utilizado por niños y niñas de educación primaria en el ámbito de sus centros de educación, bajo la supervisión y orientación de sus profesores, y en principio adaptado para su ejecución en computadores típicos de sobremesa de las aulas / laboratorios de informática, si bien se prevé la posibilidad de su uso futuro en otros dispositivos portátiles como tabletas, móviles, etc.

1.4.3. Objetivos a Nivel de Usuario

Estos objetivos de mas bajo nivel se detallan en el Modelo de Casos de Uso

1.5. Visión General del Producto

1.5.1. Perspectiva del Producto

El sistema será un entorno gráfico en el cual se podrá programar el comportamiento de uno o varios robots que se mueven por un escenario e interactúan con el. Tanto el escenario como el robot serán imágenes que se podrán cargar en el entorno, pudiendo variarse ambos para dar a los alumnos la posibilidad de crear diferentes programas.

El entorno gráfico estará compuesto por varias áreas de trabajo, entre las cuales están:

- Panel de programación del robot: área donde se podrán programar las acciones que realizará el robot en el escenario, el cual se divide en
 - Paleta de comandos: panel donde se mostrarán las todas las acciones y comandos aplicables al robot
 - Escritorio: panel donde se colocarán los comandos para crear el programa que debe ejecutar el robot
- Panel de Control: permitirá el control de la ejecución de los programas presentes en el escritorio
- Escenario: panel donde se mostrará la simulación de las acciones de los robots sobre el escenario

1.5.2. Resumen de las características del Sistema

Los requisitos funcionales del sistema se recogen en el Modelo de Casos de Uso de forma mas exhaustiva, de modo que en este apartado se hace un breve resumen las características principales:

- construcción de programas
- ejecución de programas
- parada de programas en ejecución

- reset de programas en ejecución / parados
- visualización de los diferentes tipos de comandos disponibles
- colocación de comandos en el escritorio
- eliminar un comando del escritorio
- realización de acciones de monitorización (sensores - tecla, color, etc.)
- cargar imagen como escenario
- cargar imagen como robot

1.5.3. Otros Requisitos y Restricciones

- Los requisitos no funcionales se recogen en el documento de Especificación Complementaria

2 Requisitos

2.1. Modelo de Casos de Uso

2.1.1. Introducción

Este documento trata de especificar en detalle y relacionar las funcionalidades del sistema que proporcionan valor a los usuarios, y las necesidades de usuario que cubren dichas funcionalidades.

2.1.2. Lista Evento-Actor-Objetivo

- La siguiente lista muestra los principales eventos de entrada al sistema, especificando quien es el actor que provoca tales eventos, y cual es el objetivo o causa de los mismos.
- Mediante el estudio de dichos eventos de entrada se trata de analizar las principales funcionalidades y características que debería tener el sistema, descubriéndose de este modo los principales Casos de Uso que darán forma al sistema:

Cuadro 2.1: Lista evento-actor-objetivo

Evento	nº	Actor	Objetivo
iniciarEntorno	UC0 - Iniciar Entorno	Alumno/Profesor	iniciar el Sistema para trabajar con el
colocar un comando	UC1 - Colocar comando	Alumno	realizar una acción
unir dos comandos	UC2 - Unir dos comandos	Alumno	realizar una acción a continuación de otra
mover un comando	UC3 - Mover comando	Alumno	reordenar o corregir una secuencia de acciones
registrar evento sensor	UC4 - Registrar evento sensor	Alumno	cumplir una restricción de una acción
ejecución de programas	UC5 - Ejecutar programa	Alumno	ejecutar programa / ver resultados escenario
parar programas	UC6 - Parar programa	Alumno	detener la ejecución
resetear programas	UC7 - Resetear programa	Alumno	volver a la posición inicial
cargar fondo	UC8 - Cargar fondo de escenario	Alumno	definir un fondo para la simulación
cargar robot	UC9 - Cargar robot	Alumno	definir una imagen como robot
gestionar proyectos	UC10 - Gestionar proyectos	Alumno/Profesor	guardar y cargar proyectos
hacer login	UC11 - Hacer login	Alumno/Profesor	gestión de cuentas personalizadas
limpiar escritorio	UC12 - Limpiar Escritorio	Alumno	eliminar todos los programas y comandos sueltos

2.1.3. Modelo de Casos de Uso

2.1.3.1. UC0 - Iniciar Entorno

- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere poner en funcionamiento el entorno para poder trabajar en el
- Precondiciones:
- Postcondiciones:
 - el entorno queda iniciado correctamente

- la GUI se muestra correctamente
- Escenario Principal de Éxito o Flujo Básico
 1. el Alumno inicia el Sistema
 2. el Sistema carga los elementos de la GUI
 3. el Sistema muestra la GUI
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 1a** el Sistema no se ha iniciado correctamente
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 2a** el Sistema no carga los comandos correctamente
 - las imágenes de los comandos deberán ser cargadas (a los JLabel) desde un directorio interno del proyecto, de modo que en principio no debería suceder
 - el problema aquí puede ser la especificación de la ruta
- 3a** el Sistema no muestra la GUI
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- Requisitos Especiales
 - de momento se incluye dentro de otros casos de uso, pero quizás sea mas correcto refactorizarle como UC aparte, ya que es común a varios
- Lista de tecnología
 - se usan elementos de Java Swing para la GUI
- Frecuencia
 - N/A
- Temas abiertos

2.1.3.2. UC1 - Colocar comando

- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere iniciar un programa, o bien, quiere añadir un comando al escritorio (sin acoplarle a otros) para unirla mas adelante a otro programa existente
 - en principio no se contempla en este UC el acoplamiento de comandos ya que es responsabilidad del UC2 - Unir dos comandos
- Precondiciones:
 - el Sistema se inicia correctamente
 - el comando está presente en la paleta de comandos
 - es posible, pero no necesario, que existan otros comandos en el escritorio { esto no es estrictamente una precondición pero se añade por su aporte informativo }
- Postcondiciones:
 - 1 el comando permanece en la paleta de comandos
 - 2.a el comando queda depositado en el escritorio del entorno y forma la primera sentencia del programa actual formado por el solo de momento, hasta que se le unan otros comandos, si aplica
 - es como una acción de copiar de la paleta de comandos al escritorio
 - en este caso el único comando válido sería *Start*, ya que todo programa válido debe comenzar con este comando
 - 2.b o bien, el comando queda depositado en el escritorio provisionalmente, a la espera de unirse con otros comandos de programas formados válidos
 - aquí se permite que el comando depositado en el escritorio sea cualquiera de los existentes en la paleta, ya que en principio no formará un programa válido, sino que se quedará a la espera de unirse con el ultimo comando de un programa válido preexistente
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente
 2. el Alumno elige el comando *Start* para iniciar un nuevo programa
 3. el Alumno arrastra el comando de la paleta de comandos al escritorio
 4. el Sistema mantiene el comando en la paleta de comandos
 5. el Sistema valida la colocación del comando y lo mantiene en el escritorio

6. el Sistema crea un nuevo programa en el escritorio compuesto únicamente por el comando Start colocado
 7. el Sistema muestra la GUI actualizada con el nuevo comando colocado en la posición deseada
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 1a** el Sistema no se ha iniciado correctamente
1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 2a** el Alumno elige otro comando diferente de *Start* para colocar en el escritorio
1. el Alumno arrastra el comando de la paleta de comandos al escritorio
 2. el Sistema mantiene el comando en la paleta de comandos
 3. el Sistema valida la colocación del comando y lo mantiene en el escritorio
 4. el Sistema NO crea un nuevo programa, ya que no es Start el comando, sin embargo añade este comando al escritorio para añadirle mas tarde a uno de los programas válidos
- 2b** la Paleta de comandos no contiene el comando que el alumno desea colocar
1. el Alumno puede elegir entre las siguientes opciones
 - a) el Alumno debe elegir otro comando para realizar la acción
 - b) el Alumno podría crear su propio comando personalizado para realizar la acción (posible extensión del Sistema)
- 5a** colocación de comando no válida
1. el Sistema muestra de alguna forma que la colocación no es legal
 2. el comando no se coloca en el escritorio
 3. el comando permanece en la Paleta de comandos
 4. el Alumno arrastra de nuevo el comando al escritorio
- 6a** el comando elegido NO es *Start*
1. el Sistema no crea un nuevo programa, pero añade dicho comando al escritorio de forma aislada (debería poder contener una lista de comandos sueltos para ello)
- Requisitos Especiales

- los comandos podrían estar representados por imágenes de diferentes formas y colores atendiendo a su tipo
 - se supone que en este caso de uso, se arrastra el comando al escritorio, pero no se une a ninguno existente, de modo que se crea un nuevo programa, compuesto de momento por ese único comando del escritorio que acabamos de arrastrar
- Lista de tecnología
 - los comandos estarán representados por imágenes cargadas en la paleta de comandos
 - dichas imágenes estarán contenidas en un directorio interno de la aplicación
 - se implementan como JPanels con imágenes de fondo de los comandos y JComboBoxes para los comandos que tengan elementos editables
 - los paneles involucrados estarán representados por contenedores de Java Swing
 - Frecuencia
 - el tiempo entre el arrastre de dos comandos sucesivos es de pocos segundos (no creo que tenga demasiada relevancia en este caso)
 - se ha probado en el prototipo y en el entorno de pruebas no presenta especial relevancia
 - Temas abiertos
 - Este UC se divide en extensiones en función del tipo de comando (control, sensor, actuador, etc.) - {no es necesario dividirlo en subcasos de uso, se utiliza para ello el apartado *Extensiones*}
 - se diferencia atendiendo a la *gramática del lenguaje definido*, ya que solo el comando *Start* puede ser el comando inicial de un programa válido, de modo que se diferencia entre *Start* y resto de comandos
 - mantengo el tema abierto por si necesita modificarse mas adelante
 - iniciarSistema() podría ser un UC separado (de momento lo mantenemos como una operación mas, pero ya que es común a varios tiene sentido refactorizarlo como UC)

2.1.3.3. UC2 - Unir dos comandos

- Definición: El alumno arrastra un comando de la paleta de comandos al escritorio del panel de programación (o del propio escritorio al escritorio) y acerca dicho comando a otro ya existente en el escritorio para unir ambos en dos comandos que se ejecutarán secuencialmente, y el sistema une ambos comandos quedando listos para su ejecución secuencial

- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere encadenar dos comandos que representan dos acciones que se ejecutarán secuencialmente en el programa
- Precondiciones:
 - el Sistema se inicia correctamente
 - hay al menos un comando presente en el escritorio
 - el comando que se quiere unir (comando origen), puede unirse al comando objetivo del escritorio (comando destino); existen 2 orígenes
 - paleta de comandos: se arrastra (copia) un comando desde la paleta de comandos y se une a uno existente del escritorio
 - escritorio: se mueve un comando del escritorio para acoplarlo a otro existente en el escritorio
 - el que el comando origen pueda o no unirse al comando destino depende de la definición de la gramática del lenguaje del documento *Lenguaje de Comandos* presente en el directorio de *Diseño*:
 - Start:
 - ◊ **puede unirse a:** ninguno
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
 - Repeat:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** todos excepto Start
 - a.** Sensor: por la derecha
 - b.** Resto: por debajo (entre *Repeat* y *endRepeat*)
 - Wait:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** todos excepto Start
 - a.** solo admite que se le unan comandos tipo *Sensor* dentro de el
 - b.** NOTA: admite Sensores por la derecha y todos excepto Start y Sensor por debajo (la siguiente sentencia puede ser cualquiera)
 - If:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** todos excepto Start

- a. caso idéntico al Repeat: por la derecha los tipo Sensor y por debajo el resto (Bloque de sentencias)
 - o Color:
 - ◊ **puede unirse a:** Repeat, Wait, If
 - ◊ **pueden unírsele:** ninguno
 - o Key:
 - ◊ **puede unirse a:** Repeat, Wait, If
 - ◊ **pueden unírsele:** ninguno
 - o SetPosition:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
 - o SetSpeed:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
 - o Move:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key)*
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
 - o End_Repeat:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key) y Start*
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
 - o End_If:
 - ◊ **puede unirse a:** todos excepto los tipo *Sensor (Color, Key) y Start*
 - ◊ **pueden unírsele:** Repeat, Wait, If, SetPosition, SetSpeed, Move
- Postcondiciones:
 - el comando origen permanece en la paleta de comandos
 - el comando origen permanece en el escritorio (la «copia» del comando origen de la Paleta de Comandos)
 - el comando origen queda unido al comando destino, ambos en el escritorio, formando parte de un programa en el cual las acciones representadas por ambos se ejecutarán secuencialmente (primero la del comando destino, y después la del comando origen, por ser este último colocado en último lugar)
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente

2. el Alumno elige el comando que quiere unir a otro
 3. bien la Paleta de comandos o bien el escritorio (o ambos), contienen dicho comando
 4. el Alumno acerca el comando que está arrastrando (comando origen), al comando destino, para acoplarlo a éste último (desde la paleta o desde el mismo escritorio)
 5. el Sistema mantiene el comando origen en la paleta de comandos
 6. el Sistema quizás muestra una indicación visual para notificar si ambos pueden unirse (posible extensión mas visual)
 7. el Alumno suelta el comando origen para dejarlo acoplado al comando destino
 8. el Sistema valida la colocación del comando y lo acopla al comando destino
 9. ambos comandos quedan unidos en el escritorio formando una secuencia de acciones (1º destino, 2º origen -> tal como está planteado el comando origen se une a continuación del destino)
 10. el Sistema muestra la GUI actualizada con los comandos pegados entre si
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 - 1a el Sistema no se ha iniciado correctamente
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
 - 3a la Paleta de comandos no contiene el comando buscado
 1. el Alumno puede elegir entre las siguientes opciones
 - a) el Alumno debe elegir otro comando para realizar la acción
 - b) el Alumno podría crear su propio comando personalizado para realizar la acción (posible extensión del Sistema)
 - 3b el Escritorio no contiene el comando buscado
 1. el Alumno debe colocar dicho comando en el escritorio (UC1 - Colocar comando)
 2. el Alumno puede ahora mover dicho comando desde el escritorio para acoplarlo al comando destino
 - 6a la indicación visual muestra (o no muestra) que la operación no es legal (el comando origen no se puede acoplar al comando destino)
 1. el Alumno puede optar entre
 - a) dejar el comando origen en el escritorio para usarlo mas tarde
 - b) arrastrar el comando origen fuera del escritorio para eliminarlo

2. el Alumno elige otro comando cuya unión con el comando destino sea válida y repite el proceso de acoplamiento

- Requisitos Especiales

- podría existir un indicador visual que muestre si dos comandos pueden acoplarse
 - podría ser simplemente un mensaje de dialogo del JOptionPane similar a como está ahora para el debug, pero modificando el aspecto visual
- en caso de no existir dicho indicador visual, ver como podemos dar información al Alumno para orientarle en la colocación (colores, formas de comandos, notificaciones JDialog, etc..)
- Este UC quizás deberá dividirse en casos según el tipo de comando
 - especificar que combinaciones de comandos son legales en cuanto a cuales pueden unirse a cuales
- el comando origen solo se puede acoplar al ultimo comando de un programa, excepto en caso de un Comando Sensor (Color, Key) que se puede acoplar a cualquier comando de Control con un hueco derecho libre
 - esto hace que de momento no se puedan insertar comandos entre medias de un programa (posible extensión)

- Lista de tecnología

- el indicador visual de unión podría quizás ser otro objeto de JSwing como un mensaje de diálogo del JOptionPane
- el método canSnap(Comando target) implementa las restricciones de acoplamiento de los comandos

- Frecuencia

- igual que el UC1 - Colocar comando (poco significativo en principio)

- Temas abiertos

- Este UC se divide en sub-casos en función del tipo de comando (control, sensor, actuador, etc.)
 - cada tipo de comando tendrá una semántica asociada en el sentido de que al ejecutar el programa en el entorno, cada uno motivará la realización de diferentes acciones y/o comprobaciones
- Posibilidad de acoplar comandos entre medias de un programa y no solo al final (excepto los Sensores) - posible EXTENSIÓN

2.1.3.4. UC3 - Mover comando

- NOTA: este UC se ha reformulado como «mover comando» en vez de «eliminar comando», modificando el UC2 - Unir dos comandos (para que sea solo el acto de snap, bien desde la paleta o desde el propio escritorio) , y haciendo que en función de la localización de destino del comando se haga una acción u otra
 - ver acoplamiento de este UC (donde afecta el refactorizarlo como Mover Comando)
- Definición: El Alumno mueve un comando presente en el escritorio, bien hacia otra área del mismo escritorio, o bien fuera de él.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: podría darse en principio una de las siguientes 3 motivaciones:
 - quiere cambiar la colocación de un comando dentro del escritorio por motivos de organización del espacio, por mejora de su visibilidad, o por otras razones similares
 - quiere mover un comando presente en el escritorio para acercarlo a otro comando y poder acoplar ambos
 - quiere corregir un programa del escritorio, eliminando uno de los comandos que lo forman, moviendo dicho comando fuera del escritorio
- Precondiciones:
 - el Sistema se inicia correctamente
 - hay al menos un comando presente en el escritorio
 - obviamente si el objetivo es acoplarle con otro, deberían estar presentes ambos
- Postcondiciones:
 - en función del objetivo que persiga el Alumno pueden ser:
 - el comando objetivo desaparece del escritorio, y se elimina del programa que lo contenía
 - ◊ en caso de ser el único comando del programa (*Start*) se elimina el programa del escritorio
 - el comando objetivo (comando origen) queda acoplado al comando destino (responsabilidad del UC2 - Unir dos comandos)
 - el comando objetivo permanece en el escritorio, pero su localización cambia

- ◊ si el comando que se mueve pertenecía a un programa:
 1. si el programa tenía ese único comando: solo cambia la localización de ese programa entero
 2. si el programa tenía mas comandos: el comando objetivo se elimina del programa, y se coloca el solo en el escritorio en la nueva localización
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente
 2. el Alumno elige el comando que quiere mover
 3. el escritorio contiene el comando que representa dicha acción (comando objetivo)
 4. el Alumno mueve el comando a la nueva localización (en este punto existen varios caminos detallados en EXTENSIONES)
- Extensiones o Fluxos Alternativos
 - a*** en cualquier momento el Sistema se cuelga / falla
 - 1a** el Sistema no se ha iniciado correctamente
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
 - 3a** el comando no está en el escritorio
 1. el Alumno puede:
 - a) colocar el comando desde la paleta de comandos al escritorio (UC1 - Colocar comando) y continuar desde este punto
 - b) elegir otro comando para mover y continuar
 - 4a** el Alumno quiere eliminar el comando objetivo de un programa del escritorio, dos casos
 1. único programa formado por ese comando:
 - a) el Alumno mueve el comando fuera del escritorio
 - b) el Sistema elimina el comando del programa
 - c) el Sistema elimina el programa del escritorio
 - d) el Sistema elimina el comando de la GUI
 2. programa con varios comandos:
 - a) el Alumno mueve el comando fuera del escritorio
 - b) el Sistema elimina el comando del programa
 - c) el Sistema mantiene el programa en el escritorio

1) el Sistema reordena los comandos restantes en función de la posición que ocupara el comando objetivo

2) NOTA: POSIBLE EXTENSIÓN

d) el Sistema elimina el comando de la GUI

4b el Alumno quiere mover el comando objetivo (comando origen) para acercarlo a otro existente (comando destino) y acoplarlo a este, dos casos:

1. único programa formado por ese comando:

a) el Alumno mueve el comando acercándole al comando destino para acoplarles

b) EN ESTE PUNTO SE APLICA EL UC2 - UNIR DOS COMANDOS

2. programa formado por varios comandos:

a) el Alumno mueve el comando acercándole al comando destino para acoplarles

b) el Sistema elimina el comando del programa

c) el Sistema mantiene el programa en el escritorio

1) el Sistema reordena los comandos restantes en función de la posición que ocupara el comando objetivo

2) NOTA: POSIBLE EXTENSIÓN

d) EN ESTE PUNTO SE APLICA EL UC2 - UNIR DOS COMANDOS

4c el Alumno quiere mover el comando objetivo para colocarle en otra localización del escritorio, dos casos:

1. único programa formado por ese comando:

a) el Alumno mueve el comando a otra localización del escritorio

b) el Sistema actualiza el programa con la nueva localización

2. programa con varios comandos:

a) el Alumno mueve el comando a otra localización del escritorio

b) el Sistema elimina el comando del programa

c) el Sistema mantiene el programa en el escritorio

1) el Sistema reordena los comandos restantes en función de la posición que ocupara el comando objetivo

2) NOTA: POSIBLE EXTENSIÓN

d) el Sistema añade el comando suelto al escritorio en su nueva localización

- Requisitos Especiales

- por comodidad, la papelera podría no ser un lugar específico, sino cualquier área de la GUI que no sea el escritorio (quizás mas complejo de implementar)
 - en principio se ha implementado como «fuera del escritorio»
 - los comandos tienen unos atributos booleanos que indican si forman parte de un programa o si están sueltos, en función de los cuales se permite o no el movimiento
 - de momento se ha implementado una solución en la cual los comandos acoplados a un programa se bloquean, impidiendo su movimiento o eliminación individual
- Lista de tecnología
 - la papelera podría ser un área del entorno, como un JPanel u otro contenedor específico que contenga instancias de los comandos eliminados
 - la zona de eliminación podría no ser un área específica, sino cualquier parte del entorno, excepto el escritorio
 - se opta por esta solución
 - Frecuencia
 - igual que el UC1 - Colocar comando
 - Temas abiertos
 - que debe ocurrir cuando se intenta eliminar un comando que está unido a otros ??
 - se elimina el programa entero, o solo el comando seleccionado?
 - como afecta esto a la estructura donde se almacenan los comandos (Programa) y los programas (Escritorio) ?
 - como se deben reordenar los comandos restantes del programa?
 - se deben mover varios comandos en bloque cuando se mueva alguno que no sea el último del programa?
 - es viable que el escritorio contenga una lista de comandos sueltos ademas de una colección de programas?
 - de momento no existe la posibilidad de eliminar un comando de un programa
 - POSIBLE EXTENSIÓN
 - de momento los únicos comandos que se pueden mover por el escritorio, bien sea para acoplarlos a otros, para eliminarlos del escritorio o para reorganizarlos, son aquellos comandos que están sueltos, sin formar parte de un programa
 - POSIBLE EXTENSIÓN

2.1.3.5. UC4 - Registrar evento sensor

- Definición: El Sistema registra en indicadores especiales del Panel de Control los eventos que sirven como sensores para las Sentencias de Control de los programas del Escritorio
- Actor Principal: Alumno / Sistema
- Personal Involucrado e Intereses:
 - Alumno: quiere dejar registrado un evento de entrada de tecla presionada
 - Sistema: quiere dejar registrado el color actual del Fondo donde esta posicionado el Robot en el Escenario
- Precondiciones:
 - en principio no es necesaria ninguna precondición especial adicional al inicio del Sistema
- Postcondiciones:
 - el indicador en cuestión refleja la información pertinente correctamente
- Escenario Principal de Éxito o Flujo Básico (4 casos)
 1. Evento Teclado
 - a) el Alumno presiona una tecla
 - b) el Panel de Control oye el evento del teclado
 - c) el Sistema registra la tecla presionada
 - d) el Panel de Control actualiza la información de su componente «última tecla»
 2. Evento Color
 - a) el Robot está posicionado sobre el fondo del Escenario
 - b) el Sistema registra el color de la casilla (pixel) de la imagen de fondo sobre la cual esta el Robot
 - c) el Panel de Control actualiza la información de su componente «color actual»
 3. Evento Posición Ratón
 - a) el Alumno hace click en el Escenario
 - b) el Panel de Control oye el evento del ratón
 - c) el Sistema registra la posición del cursor en el Escenario
 - d) el Panel de Control actualiza la información de su componente «posición ratón»

4. Evento Color Ratón

- a) el Alumno hace click en el Escenario
- b) el Panel de Control oye el evento del ratón
- c) el Sistema registra el color de la posición del cursor en el Escenario
- d) el Panel de Control actualiza la información de su componente «color ratón»

■ Extensiones o Fluxos Alternativos

a* en cualquier momento el Sistema se cuelga / falla

- 1. el Alumno / Profesor cierra el Sistema
- 2. el Alumno / Profesor reinicia el Sistema

1(a)a el Alumno no ha presionado aun ninguna tecla

- 1. el sensor ultima tecla debería contener un valor inicial por defecto

2(a)a el Robot aun no se ha movido en el escenario (está en su posición inicial)

- 1. el sensor color actual debería contener un valor inicial (quizás el valor actual real de la posición inicial del fondo actual)

3(a)a el Alumno no ha hecho click aún en el Escenario

- 1. el sensor posición ratón debería contener un valor inicial por defecto (mensaje «coords» p.e.)

4(a)a el Alumno no ha hecho click aún en el Escenario

- 1. el sensor color ratón debería contener un valor inicial por defecto (blanco p.e.)

■ Requisitos Especiales

- ver que cantidad de colores es necesario reconocer
 - está en función del numero de imágenes de color que se incluyan en el ComandoColor
 - cambiar la implementación de los colores de ComandoColor para que fueran directamente colores (clase Color java)
- quizás sea necesario inicializar estos sensores con valores por defecto

■ Lista de tecnología

- uso de JLabels con fondos de color y mensajes de texto
- uso de la clase Color de java
- quizás sea necesario extender esta clase con nuevas funcionalidades

■ Frecuencia

- podría ser casi constante, tanto para el caso de teclado como el caso del color del fondo
 - teclado: el Alumno no debería tener que esperar un tiempo significativo entre cada pulsación de tecla
 - color: el robot puede estar moviéndose rápidamente por el escenario, y en cada «paso» hay que registrar el color actual
- Temas abiertos
 - como determinamos el color de la posición actual del Robot ?
 - la posición del robot se define obteniendo el pixel central de la casilla que ocupa el Robot en cada momento
 - valorar el uso de items de colores en cada mapa para el uso de los sensores de color (pintando cada casilla del color de esos items específicos) - posible extensión

2.1.3.6. UC5 - Ejecutar programa

- Definición: El alumno hace click en el botón correspondiente del panel de control para ejecutar los programas actuales existentes en el escritorio del entorno. El sistema ejecuta en ese momento todos los programas presentes en el escritorio.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere ejecutar las acciones programadas y definidas por los programas que ha creado y colocado en el escritorio
 - en ultima instancia quiere que el robot presente en el escenario realice una serie de acciones sobre el fondo de escenario
- Precondiciones:
 - el Sistema se inicia correctamente
 - hay al menos un programa presente en el escritorio
- Postcondiciones:
 - el robot ejecuta la secuencia de acciones definidas por los programas del escritorio
 - la secuencia de acciones que ejecuta el robot sobre el fondo se ajusta a lo que el Alumno desea que el robot haga (especificaciones del Alumno)
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente

2. el Sistema muestra la GUI
 3. el Alumno ha creado correctamente uno o varios programas en el escritorio
 4. el Alumno presiona el botón Start del panel de control
 5. el Sistema ejecuta secuencialmente las acciones definidas por los programas del escritorio
 6. el Sistema sigue ejecutándose hasta que el Alumno presione el botón Stop del panel de control (UC6 - Parar programa) o hasta que terminan todas las acciones de los programas, o hasta que se llega fuera de los límites válidos
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
- 1a** el Sistema no se ha iniciado correctamente
1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 2a** la GUI no se muestra adecuadamente
1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 3a** no existe ningún programa válido en el escritorio
1. el Alumno puede elegir entre:
 - a) no hacer nada mas, con lo que finaliza la ejecución y el UC
 - b) crear y colocar un nuevo programa en el escritorio (UC1 - Colocar comando, UC2 - Unir dos comandos, UC3 - Mover comando) y continuar desde este punto
- 4a** el Alumno pulsa varias veces seguidas el botón Start
1. el efecto debería ser como si solo se pulsara una vez el botón - pendiente
- 5a** el Sistema no ejecuta las acciones de los programas válidos después de presionar el botón Start
1. NOTA: en principio no debería ser una extensión de este UC (ver como se procedería en este caso)
- Requisitos Especiales
 - el botón Start debería ser lo suficientemente intuitivo para que los alumnos entiendan su propósito
 - se deben ejecutar simultáneamente TODOS los programas presentes en el escritorio
 - esto implica en principio la gestión de varios hilos, 1 por programa (control de concurrencia)

- Lista de tecnología
 - en principio se usarán JButtons de java swing
- Frecuencia
 - en principio solo una ejecución puede estar activa al mismo tiempo
 - un hilo por programa
 - hasta que termine el hilo no se pueden crear otro hilo para ese programa
 - ver relación con multithreading
- Temas abiertos
 - Conurrencia:
 - ejecución de varios programas al tiempo
 - uso de varios hilos, 1 por cada programa, mas el hilo principal de ejecución del Entorno (java.concurrency: uso de Executors, Threads, Runnables, paso de mensajes ...)
 - 1 hilo por programa en cada momento
 - posibilidad de tener un escritorio diferente para cada robot / objeto - POSIBLE EXTENSIÓN
 - Este UC deberá dividirse en casos según el tipo de comando ?
 - si el Alumno pulsa varias veces seguidas el botón Start el efecto debería ser como si solo se pulsara una vez el botón

2.1.3.7. UC6 - Parar programa

- Definición: El alumno hace click en el botón correspondiente del panel de control para detener los programas actualmente en ejecución presentes en el escritorio del entorno. El sistema detiene todos los programas actualmente en ejecución.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere detener la ejecución de los programas del escritorio actualmente en ejecución
- Precondiciones:
 - el Sistema se inicia correctamente
 - hay al menos un programa en ejecución presente en el escritorio
- Postcondiciones:
 - todos los programas del escritorio han visto detenida su ejecución

- Escenario Principal de Éxito o Flujo Básico
 - NOTA: este UC sería la continuación del flujo básico del UC5 - Ejecutar programa
 1. el Sistema tiene uno o varios programas en ejecución
 2. el Alumno presiona el botón Stop del panel de control
 3. el Sistema detiene la ejecución de todos los programas
- Extensiones o Flujos Alternativos
 - NOTA: las extensiones hasta el momento de la puesta en ejecución de los programas serían las mismas que en el UC anterior
- a* en cualquier momento el Sistema se cuelga / falla
 - 1. el Alumno / Profesor cierra el Sistema
 - 2. el Alumno / Profesor reinicia el Sistema
- 1a el Sistema no tiene ningún programa en ejecución, dos casos:
 - a. si no se hacen mas acciones, el UC termina
 - b. si se desea continuar habrá que volver atrás al UC5 - Ejecutar programa y crear y ejecutar uno o varios programas
- 3a la ejecución de los programas no se detiene
 - 1. no debería ocurrir, pero en su caso se procederá como si el Sistema se hubiese colgado
- Requisitos Especiales
 - el botón Stop debería ser lo suficientemente intuitivo para que los alumnos entiendan su propósito
 - se deben detener simultáneamente TODOS los programas presentes en el escritorio
 - el botón Stop define 2 estados del Panel de Control, en función de su atributo stopped, parado (stopped) y listo (!stopped), y cada pulsación del botón cambia de estado, prohibiendo o permitiendo la ejecución de los programas
- Lista de tecnología
 - en principio se usarán JButtons de java swing
- Frecuencia
 - N/A
- Temas abiertos

2.1.3.8. UC7 - Resetear programa

- Definición: El alumno hace click en el botón correspondiente del panel de control para devolver el escenario a su estado inicial, reiniciando y deteniendo los programas actualmente en ejecución del escritorio. El sistema revierte el escenario a su estado inicial, recuperando los valores iniciales del escenario y de los robots.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere volver a la posición inicial de los robots en el escenario (quizás mas preciso sea decir «estado inicial» ya que habrá que resetear varias características ...)
- Precondiciones:
 - el Sistema se inicia correctamente
 - todos los programas del escritorio están detenidos
- Postcondiciones:
 - el robot del escenario vuelve a su posición inicial
- Escenario Principal de Éxito o Flujo Básico
 - NOTA: este UC sería la continuación del flujo básico del UC6 - Parar programa
 1. el Sistema tiene uno o varios programas cuya ejecución se ha detenido
 2. el Alumno presiona el botón Reset del panel de control
 3. el Sistema devuelve al robot a su posición inicial en el escenario
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 1a** el Sistema no tiene ningún programa cuya ejecución se haya detenido:
 1. el Alumno debe presionar el botón Stop para detener la ejecución de los programas
- 3a** el Sistema no devuelve al robot a su posición inicial
 1. se procederá como si el Sistema se hubiera colgado
- Requisitos Especiales
 - el botón Reset debería ser lo suficientemente intuitivo para que los alumnos entiendan su propósito

- solo se podrá resetear en el estado de parada del Panel de Control
 - el robot quizás tenga mas características ademas de su posición en el fondo del escenario, como su color, orientación, velocidad, etc. en cuyo caso habrá que devolver todas ellas a su valor inicial - pendiente
- Lista de tecnología
 - en principio se usarán JButtons de java swing
 - Frecuencia
 - N/A
 - Temas abiertos
 - for (Feature f: robot.getFeature()) { robot.setDefault(f) }
 - ver si esto implica la creación de una FSM que guarde y actualice el estado del sistema

2.1.3.9. UC8 - Cargar fondo de escenario

- Definición: El alumno elige una imagen que sirva de fondo del escenario y la carga en el sistema para que se muestre en el panel de escenario. El sistema verifica que la imagen es adecuada y la carga en el escenario como fondo.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere establecer un fondo para el escenario donde se moverá el robot
- Precondiciones:
 - el Sistema se inicia correctamente
 - existen imágenes de fondo válidas en una carpeta interna del sistema
 - nota: quizás se podría permitir que se cargaran imágenes de cualquier otra fuente
- Postcondiciones:
 - el fondo del escenario queda fijado en el escenario
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente
 2. el Sistema muestra la GUI
 3. el Alumno selecciona la opción de cargar fondo desde la GUI

4. el Alumno selecciona un fondo válido para el escenario del directorio de imágenes
 5. el Sistema establece la imagen seleccionada como fondo de escritorio
- Extensiones o Fluxos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 1a** el Sistema no muestra la GUI
1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 4a** el Alumno selecciona una imagen desde otra fuente (Internet, disco local, etc...)
1. se debería proceder análogamente al flujo básico
- 5a** el Sistema no muestra la imagen de fondo correctamente, dos opciones:
- a. el Alumno / Profesor reinicia el Sistema
 - b. el Alumno selecciona otra imagen como fondo
- Requisitos Especiales
 - la opción de cargar fondo, debería estar presente en la GUI, bien como botón o como elemento de menú (por ejemplo)
 - se prevé la posibilidad de cargar imágenes desde diferentes fuentes, aparte del directorio de imágenes del Sistema
 - el uso de imágenes predefinidas para la aplicación o no, da lugar a dos tipos de modos de uso del Sistema:
 - mapas cerrados: imágenes predefinidas que crean mapas con elementos interactivos, donde el Robot puede seguir un camino para completar el mapa
 - mapas abiertos: cualquier otra imagen, usados para practicar con diferentes programas y experimentar con las acciones del Robot
- Lista de tecnología
 - se podrían usar jButtons, JMenu, etc.
 - JFileChooser
 - Frecuencia
 - N/A
 - Temas abiertos

- formatos de imagen válidos
 - que tipo de archivos debe aceptar (jpg, png ...)?
 - que dimensiones de imagen son válidas (afecta al layout del Entorno)?

2.1.3.10. UC9 - Cargar robot

- Definición: El alumno elige una imagen que sirva de robot móvil e interactivo del escenario, para el cual se programarán las acciones que realizará según el programa creado y presente en el escritorio. El sistema valida la imagen como robot y la carga en una posición por defecto en el escenario.
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere establecer una imagen como el robot interactivo que se moverá por el escenario en respuesta a las acciones programadas
- Precondiciones:
 - el Sistema se inicia correctamente
 - existen imágenes de robot válidas en una carpeta interna del sistema
 - nota: quizás se podría permitir que se cargaran imágenes de cualquier otra fuente
- Postcondiciones:
 - el robot queda fijado en el escenario
- Escenario Principal de Éxito o Flujo Básico
 1. el Sistema se ha iniciado correctamente
 2. el Sistema muestra la GUI
 3. el Alumno selecciona la opción de cargar robot desde la GUI
 4. el Alumno selecciona una imagen válida para el robot del directorio de imágenes
 5. el Sistema establece la imagen seleccionada como robot
- Extensiones o Flujos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
 - 1a el Sistema no muestra la GUI

1. el Alumno / Profesor cierra el Sistema
2. el Alumno / Profesor reinicia el Sistema

4a el Alumno selecciona una imagen desde otra fuente (Internet, disco local, etc...)

- NOTA: ver que se entiende por imagen válida
- 1. se debería proceder análogamente al flujo básico

5a el Sistema no muestra la imagen del robot correctamente, dos opciones:

- a.** el Alumno / Profesor reinicia el Sistema
- b.** el Alumno selecciona otra imagen como robot

- Requisitos Especiales

- la opción de cargar robot, debería estar presente en la GUI, bien como botón o como elemento de menú (por ejemplo)
- se prevé la posibilidad de cargar imágenes desde diferentes fuentes, aparte del directorio de imágenes del Sistema
- se prevé la posibilidad de usar varios robots que se muevan por el escenario
 - en este caso, podría ser interesante que existieran diferentes escritorios, cada uno asociados a un robot (quizás como TABS de un JTabbedPane)

- Lista de tecnología

- se podrían usar JButton, JMenu, etc. para la opción de cargar robot de la GUI
- en caso de existir varios robots, y asociar un escritorio a cada uno, se podrían usar los tabs de un JTabbedPane (escritorio)
- JFileChooser

- Frecuencia

- N/A

- Temas abiertos

- formatos de imagen válidos
 - que tipo de archivos debe aceptar (jpg, png, gif ...)?
 - que dimensiones de imagen son válidas (afecta al layout del Entorno)?
- cargar mas de un robot
 - un escenario asociado a cada robot
- orientación del robot
- crear personalizados (imágenes propias)
- cargar desde diferentes fuentes

2.1.3.11. UC10 - Gestionar proyectos

- Definición: El alumno desea realizar una de la siguientes acciones, seleccionando la opción de menú correspondiente:
 - Crear un nuevo proyecto
 - Guardar el proyecto en curso
 - Cargar un proyecto guardado previamente
 - Eliminar un proyecto guardado previamente
- Temas abiertos
 - pendiente de implementar este UC - POSIBLE EXTENSIÓN

2.1.3.12. UC11 - Hacer login

- Definición: el alumno desea identificarse ante el sistema, para poder acceder a su cuenta de usuario personalizada y acceder sus proyectos guardados. El sistema verifica el ID del alumno y abre su sesión de usuario.
- Temas abiertos
 - pendiente de implementar este UC - POSIBLE EXTENSIÓN

2.1.3.13. UC12 - Limpiar Escritorio

- Definición: El alumno hace click en el botón correspondiente del panel de control para eliminar todos los programas y comandos sueltos (tanto a nivel gráfico como lógico) del Escritorio
- Actor Principal: Alumno
- Personal Involucrado e Intereses:
 - Alumno: quiere corregir algún programa o limpiar el Escritorio para crear programas nuevos
- Precondiciones:
 - el Sistema se inicia correctamente
 - existen uno o mas programas y/o comandos sueltos en el Escritorio
- Postcondiciones:
 - el Escritorio queda en blanco, habiéndose eliminado todos los programas y comandos de él.
- Escenario Principal de Éxito o Flujo Básico

1. el Alumno presiona el botón Clear del panel de control
 2. el Sistema elimina todos los comandos y programas (tanto a nivel gráfico como lógico) del Escritorio
- Extensiones o Flujos Alternativos
 - a* en cualquier momento el Sistema se cuelga / falla
 1. el Alumno / Profesor cierra el Sistema
 2. el Alumno / Profesor reinicia el Sistema
- 2a** el Sistema no elimina todos los programas / comandos (no debería ocurrir)
1. se procederá como si el Sistema se hubiera colgado
- Requisitos Especiales
 - el botón Clear debería ser lo suficientemente intuitivo para que los alumnos entiendan su propósito
 - Lista de tecnología
 - en principio se usarán JButtons de java swing
 - Frecuencia
 - N/A
 - Temas abiertos
 - pendiente de implementar como extensión una solución menos drástica de eliminación de comandos - POSIBLE EXTENSIÓN

2.2. Contratos de las operaciones

2.2.1. Introducción

En este documento se detallan aquellas operaciones que debe realizar el sistema, en respuesta a los eventos de los DSS, especificando los efectos que tienen sobre el modelo de dominio, sus clases, asociaciones y atributos

Se detallan aquellas operaciones complejas que no quedan especificadas claramente en el Modelo de Casos de Uso, y que por su relevancia merecen una especial atención y detalle, sin embargo se detallan a nivel de requisitos, sin entrar en detalles de implementación

2.2.2. Contratos

2.2.2.1. CO1: Iniciar Entorno

Operación: iniciarEntorno()

Referencias_cruzadas:

- UC0 - Iniciar Entorno
- UC1 - Colocar comando
- UC2 - Unir dos comandos
- UC3 - Mover comando
- UC11 - Hacer login (quizás)

Precondiciones: N/A

Postcondiciones:

- se creó una nueva instancia del Entorno, e (creación / destrucción de instancias)
- se creó una nueva instancia de Panel de Control, pc
- e asoció pc con e (creación / rotura de asociaciones)
- se cargaron todos los botones e indicadores disponibles
 - se creó una instancia de cada tipo de botón, b1, b2, .. bn
 - se iniciaron los colores, formas y atributos de los botones a sus valores iniciales
 - se asociaron todos los botones [b1, b2 ... bn] con pc
- se creó una nueva instancia de Paleta de Comandos, p
- se asoció p con e (creación / rotura de asociaciones)
- se cargaron todos los comandos disponibles
 - se creó una instancia de cada tipo de comando, c1, c2, .. cn
 - se iniciaron los colores y formas de los comandos a sus valores por defecto (modificación de atts)
 - se asociaron todos los comandos [c1, c2 ... cn] con p
- se creó una nueva instancia de Escritorio, et
- se asoció et con e
- se creó una nueva instancia de Escenario, en
- se asoció en con e

Diagrama_de_Interacción: DI iniciarEntorno()

2.2.2.2. CO2: Arrastrar Comando

Operación: arrastrarComando(comando: Comando)

Descripción:

- esta operación consiste en la acción del drag dentro de la mecánica del Drag and Drop
- se instrumenta a través del movimiento de la imagen fantasma del comando por el Entorno

Referencias_cruzadas:

- UC1 - Colocar comando
- UC2 - Unir dos comandos
- UC3 - Mover comando

Precondiciones:

- se ha creado una instancia de Paleta de Comandos, p
- se ha creado una instancia de Escritorio, e
- la paleta de comandos contiene dicho comando, c

Postcondiciones:

- se creó una instancia de Comando c', por copia de c
- la imagen de c' se mueve por el Entorno arrastrada por evento de mouseDragged
- la imagen de c' cambia de posición según se va moviendo por el Entorno

Diagrama_de_Interacción:DI arrastrarComando()

2.2.2.3. CO3: Acoplar Comando

Operación: acoplarComando(Comando comO, Comando comD)

- comO: el comando (comando origen) que vamos a acoplar a otro existente en el escritorio (comando destino)
- comD: el comando destino previamente existente en el escritorio

Descripción:

- esta operación consiste en la acción del snap propiamente dicho, es decir, el acoplamiento de dos comandos en el Escritorio
- el comando origen puede tener su origen en la Paleta o en el Escritorio

Referencias_cruzadas:

- UC2 - Unir dos comandos

Precondiciones:

- el Sistema se inició correctamente
- el comando origen existe bien en la Paleta de comandos, o bien en el Escritorio
- el comando destino existe en el Escritorio, comD
- existe al menos una instancia de Programa existente en el Escritorio, pr, a la cual pertenece comD

Postcondiciones:

- si el comO se arrastra desde la Paleta
 - se creó una nueva instancia del Comando origen, cO
- si no, (si se movió desde el Escritorio)
 - no se crea ninguna nueva instancia de comando
- el comando Origen se asoció con pr

Diagrama_de_Interacción:DI snap/place()**2.2.2.4. CO4: Eliminar Comando****Operación:** eliminarComando(Comando com)

- com: el comando objetivo que vamos a eliminar de un programa existente en el Escritorio

Referencias_cruzadas:

- UC3 - Mover comando

Precondiciones:

- el Sistema se inició correctamente (esta es una precondición trivial, plantearse eliminarla de los Contratos / UC donde no sea relevante)

- el comando objetivo, com, está presente en el Escritorio. 2 casos:
 - está él solo, sin estar unido a otros
 - forma parte de un programa con mas comandos (está unido a otros)

Postcondiciones:

- caso 1: forma parte de un programa con mas comandos
 - se eliminó com de pr (se elimina el com de la lista de comandos del Programa pr)
- caso 2: forma un programa él solo (no está unido a otros)
 - se eliminó com de pr
 - se eliminó pr del Escritorio e (de esta instancia de escritorio, e, si vamos a implementar mas de uno)

NOTA: de momento la eliminación de comandos del Escritorio se limita a comandos sueltos (not snapped), para la eliminación de programas se requiere limpiar todo el Escritorio, con el botón Clear del Panel de Control

Diagrama_de_Interacción: N/A

2.2.2.5. CO5: Colocar Comando

Operación: colocarComando(punto, comandoOrigen)

- comandoOrigen: el comando que vamos a añadir en el Escritorio

Descripción:

- esta operación representa la acción del drop del DnD, mediante la cual se coloca un nuevo comando en el Escritorio
- no se contempla el snap en esta operación ya que es responsabilidad del UC2 - Unir dos comandos y del CO3: Acoplar Comando

Referencias_cruzadas:

- UC1 - Colocar comando

Precondiciones:

- se ha iniciado correctamente el Entorno
- se ha recibido el evento del drop

Postcondiciones:

- se ha creado un nuevo comando c', por copia del Comando c, original de la Paleta de Comandos
- c' se ha depositado en el Escritorio, e
- si c' es START
 - se ha formado un nuevo programa p
 - c' se ha asociado con p
 - p se ha asociado con e
- si c' no es START
 - c' se ha asociado con e

Diagrama_de_Interacción:DI snap/place()**2.2.2.6. CO6: Elegir Comando****Operación:** elegirComando(comandoOrigen)

- comandoOrigen: el comando que vamos a arrastrar

Descripción:

- esta operación representa la acción del click inicial antes del drag del DnD

Referencias_cruzadas:

- UC1 - Colocar comando
- UC2 - Unir dos comandos
- UC3 - Mover comando

Precondiciones:

- se ha iniciado correctamente el Entorno
- existe una instancia de Paleta de Comandos (está implícito en el anterior requisito)
- el comando objetivo, c, existe en la Paleta

Postcondiciones:

- se ha creado un nuevo comando c', por copia del Comando c, original de la Paleta de Comandos
- c' se ha pintado en pantalla por encima del Entorno (c' se asocia con un glassPane)

Diagrama_de_Interacción:DI elegirComando()

2.2.2.7. CO7: Mover Comando

Operación: moverComando(comandoOrigen)

- comandoOrigen: el comando que vamos a mover desde el Escritorio

Descripción:

- esta operación representa la acción de mover un comando desde el Escritorio para eliminarlo, acoplarlo a otro o recolocarlo, tal como aparece en el UC3 - Mover comando

Referencias_cruzadas:

- UC3 - Mover comando

Precondiciones:

- el comando Origen c, existe en el Escritorio, e
- 2 casos
 - el comando está suelto, sin unirse a otros
 - el comando está acoplado a otros, formando parte de un programa (POSIBLE EXTENSIÓN)

Postcondiciones:

- 3 casos según se exponen en el UC3 - Mover comando
 - eliminación:
 - c se elimina del Escritorio
 - snap
 - c se acopla a otro comando c'
 - c' pertenece a un programa p
 - c se asocia con p
 - se actualizan las coordenadas gráficas de c
 - reorganización
 - se actualizan las coordenadas gráficas de c

Diagrama_de_Interacción: DI moverComando()

2.3. Especificación Complementaria

2.3.1. Introducción

En este documento se recogen los principales requisitos no funcionales, que por tanto no están definidos en el Modelo de Casos de Uso, así como un resumen de las principales funcionalidades del sistema comunes a la mayoría de casos de uso.

2.3.2. Funcionalidades

Requisitos comunes a los UCs

- ejecución de programas
 - debe desglosarse en función del tipo de comandos (lenguaje de comandos de alto nivel)
- parada de programas en ejecución
- reset de programas en ejecución / parados
- visualización de los diferentes tipos de comandos disponibles
- colocación de comandos en el escritorio
 - debe desglosarse en función del tipo de comandos (lenguaje de comandos de alto nivel)
- eliminar un comando del escritorio
- realizacion de acciones de monitorizacion (sensores - tecla, color, etc.)
- cargar imagen como escenario
- cargar imagen como robot
- gestionar proyectos
- login (gestión de cuentas de usuario)

2.3.3. Usabilidad

- Los usuarios principales van a ser niños de primaria, por lo cual el entorno gráfico con el cual van a interactuar debe ser lo mas intuitivo posible
- Se prevé la posibilidad de usar diferentes colores para los comandos para una mejor identificación de las categorías de comandos por los alumnos
- La distribución de los paneles que componen el entorno gráfico debe ser intuitiva y sencilla de entender para niños de edades entre 6-12 años

- El tamaño del escenario podría adaptarse para pantalla completa para mejorar su visualización
- La curva de aprendizaje del sistema debería tener una pendiente suave, ya que los usuarios del sistema van a ser principalmente niños de primaria, y si ellos no ven progresos rápidos en su aprendizaje esto puede hacerlos perder el interés rápidamente y desmotivarles
 - Se prevé la posibilidad de que existan tutoriales con proyectos de ejemplo para instruir a los alumnos en el funcionamiento general del sistema
 - El entorno debería contar con una función de ayuda que explique claramente como operar en el
 - Se prevé la posibilidad de que exista un pequeño vídeo introductorio que haga las veces de manual de usuario, quizás complementando una versión escrita del mismo
- localización:
 - en principio tanto los menús del sistema como los comandos se expresarán en Español, así como el manual de usuario y el resto de documentación
 - se contempla la posibilidad de adaptarlo en el futuro a otros idiomas, de modo que debería preverse esta posibilidad para facilitar su adaptación

2.3.4. Fiabilidad

- Recuperación ante caídas / errores / crashes
 - posibilidad de revertir el sistema a un estado anterior seguro, en caso de caída, para no perder el proyecto en curso

2.3.5. Rendimiento

- velocidad de ejecución de los programas
 - debería existir una velocidad por defecto y la posibilidad de variarla, para comprobaciones, etc.
- se prevé la posibilidad de interacción directa con robot (mediante click en escenario)
 - en este caso debería existir una respuesta rápida (evitar latencia), para que se perciba fluidez en la interacción
- los tiempos de inicio del sistema (startup time) y cierre (shutdown time) deben ser razonablemente reducidos

2.3.6. Soporte

2.3.6.1. Implementación

- se prevé usar tecnología Java, para facilitar la portabilidad y soporte, y facilitar el desarrollo OO, y java Swings para la creación de la GUI
- se contempla el uso de componentes Java de libre distribución para el desarrollo del sistema
- aunque en este punto aún es prematuro especificar la arquitectura del sistema, por el tipo de sistema previsto, se contempla una estructuración del sistema en capas, donde podrían existir 3 niveles:
 - capa de interfaz
 - capa de lógica del dominio
 - capa de soporte
- se prevé la posibilidad de implementar un lenguaje de programación cuya sintaxis sean los propios comandos, y cuya semántica venga definida por las acciones que realiza cada uno
- que objetos del Modelo de Diseño serán implementados en Swing directamente (capa GUI) y cuales no (capa dominio) ?
- el Escenario podría ser una matriz de posiciones bidimensional (grid)

2.3.6.2. Instalación

- el sistema debe ser de instalación sencilla en la mayoría de computadores y sistemas operativos

2.3.6.3. Escalabilidad

- Ampliar con nuevos botones / comandos
- Posibilidad de definir nuevos comandos (definidos por el usuario)
- Posibilidad de crear variables y estructuras complejas de datos
- Se prevé la posibilidad de adaptarlo para su accesibilidad a un público objetivo más amplio
- Se contempla la posibilidad de que existan varios robots en el escenario

2.3.6.4. Configurabilidad

- el sistema debería tener ciertas opciones de configuración personalizadas para adaptarse a las necesidades individuales de cada usuario

2.3.6.5. Compatibilidad

- en principio la aplicación no debería consumir demasiados recursos de procesamiento, de modo que debería ser compatible con cualquier hardware ordinario
- debería ser compatible con los principales SO

2.3.7. Interfaces

2.3.7.1. Interfaces de Usuario

Descripción La interfaz de usuario será un entorno gráfico en el cual se podrá programar el comportamiento de uno o varios robots que se mueven por un escenario e interactúan con él. Tanto el fondo del escenario como el robot serán imágenes que se podrán cargar en el entorno, pudiendo variarse ambos para dar a los alumnos la posibilidad de crear diferentes programas.

El entorno gráfico estará compuesto por varias áreas de trabajo, entre las cuales están:

- Panel de programación del robot: área donde se podrán programar las acciones que realizará el robot en el escenario
- Panel de Control: permitirá el control de la ejecución de los programas presentes en el escritorio
- Escenario: panel donde se mostrará la simulación de las acciones de los robots sobre el escenario

Disposición y Navegación La interfaz de usuario estará compuesta por 3 áreas diferenciadas, correspondientes a los siguientes 3 paneles:

- Panel de Programación: en este área es donde los alumnos crearán los programas ejecutables, la cual se subdivide en 2 subpaneles:
 - Paleta de comandos: se trata de uno o varios paneles (quizás clasificados por tipo de comando) donde se podrán visualizar todos los comandos disponibles, representados mediante imágenes de botones o bloques de diferentes colores y/o formas, los cuales pueden utilizarse y combinarse entre si para crear los programas. Existirán al menos 3 categorías de comandos disponibles para su uso:
 - Comandos de Control:
 - ◊ Start: comienzo de programa.
 - ◊ Repeat: bucle estilo “repeat S until C”.
 - ◊ Wait: espera que se produzca una condición (que uno de los sensores tome el valor esperado).
 - Sensores:

- ◊ Color: comprueba si en la posición del escenario donde se encuentra un robot aparece un determinado color.
- ◊ Key: comprueba si se ha pulsado una determinada tecla.
- Actuadores:
 - ◊ SetPosition: determina coordenadas X e Y donde posicionar el robot. Por defecto (0,0), que corresponderán al punto central del escenario
 - ◊ SetSpeed: determina la velocidad de desplazamiento del robot.
 - ◊ Move: desplaza el robot un paso en la dirección de movimiento indicada.
- Escritorio: área donde se pueden colocar los comandos para formar los programas, los cuales estarán representados por secuencias de imágenes de comando pegadas entre si. Podrán existir varios programas en este panel, pudiéndose ejecutar todos ellos a la vez, mediante el botón correspondiente del panel de control.
- Panel de Control: panel mediante el cual se podrá controlar la ejecución de los programas presentes en el escritorio del panel de programación. Contendrá al menos los siguientes botones:
 - Start: al pulsar este botón se ejecutan todos los programas presentes en el escritorio
 - Stop: detiene la ejecución de los programas actualmente en ejecución
 - Reset: restablece al estado inicial los robots (objetos) y el escenario
 - Clear: elimina todos los programas del Escritorio
 - Ultima Tecla: sensor que contiene información sobre la última tecla pulsada
 - Color: sensor que contiene información sobre el color sobre el cual está posicionado actualmente el / los robots
 - Posición Ratón: permite ver las coordenadas donde se ha hecho click en el Escenario
 - Color Ratón: permite ver el color del pixel correspondiente a las coordenadas donde se ha hecho click en el Escenario
- Escenario: panel donde se pueden visualizar las acciones que ejecutan los robots sobre el fondo del escenario, las cuales han sido definidas por los programas creados y en ejecución

Consistencia

- los paneles deben poder cambiarse de tamaño, manteniéndose unidos al resto, sin que el entorno pierda su disposición (layout) actual

- quizás no todos los comandos puedan unirse o aceptar que otros comandos se les unan, esto dependerá del tipo de comando considerado. Si son susceptibles de unirse entre si debería indicarse de alguna forma visual para que los alumnos se percaten de ello fácilmente (usabilidad)
- arrastre de comandos:
 - los comandos deben poder arrastrarse de la paleta de comandos al escritorio para constituir parte de un programa o varios.
 - del mismo modo para eliminar un comando, se le debe poder arrastrar del escritorio a alguna zona designada para su eliminación.
 - cualquier otra maniobra de arrastre de un comando se considerará no válida y no producirá efecto alguno
- quizás se deba asociar un panel de escritorio con un robot determinado (scratch), para posibilitar la existencia de varios robots, cada uno de los cuales tiene sus propios programas en el escritorio

Personalización

- se prevé la posibilidad de que el usuario pueda crear comandos personalizados
- posibilidad de adaptar los colores de los comandos y paneles al gusto del usuario
- posibilidad de cambiar la disposición de los paneles según las necesidades del usuario

2.3.7.2. Interfaces con Sistemas Externos

- en principio no se prevé la interacción con sistemas externos, pero cabe un análisis posterior de este aspecto en futuras fases del proyecto.

2.3.8. Aspectos Legales

- no se prevén implicaciones legales significativas en principio, pero se contempla un posterior análisis mas en detalle de este tema.

2.4. Glosario

2.4.1. Introducción

En este documento se recoge una relación de términos usados en el desarrollo del sistema, que identifican diversas características o componentes del mismo, y los cuales, para evitar interpretaciones ambiguas, son definidos formalmente aquí, para dejar claro su significado al personal involucrado.

Cuadro 2.2: Glosario

Término	Descripción	Alias
Paleta de comandos	parte del panel de programación donde se muestran los comandos disponibles	
Escritorio	parte del panel de programación donde se colocan los comandos para formar programas	
Escenario	panel donde se visualiza la ejecución de los programas	
Comando	cada una de las sentencias del lenguaje de programación de alto nivel utilizables, representadas mediante imágenes de etiquetas con diferentes formas y colores	Bloque / Etiqueta
Comando gráfico	imágenes manipulables de la GUI	
Comando lógico	tokens de la gramática que define la lógica de la aplicación	
Botón	botones del panel de control que controlan la ejecución de los programas	
Acción	acciones realizables en los programas	
Zona de eliminación	zona del entorno donde se colocarán los comandos para eliminarlos del escritorio	Papelera
Programa	cada uno de los programas presentes en el escritorio, representado por secuencias de comandos pegados entre si	
Robot	objeto interactivo cuyas acciones se visualizan en el escenario, representado por una imagen o una animación	
Fondo	imagen que sirve de fondo de escenario donde se moverá el robot	Mapa
Acoplar	acción de unir dos comandos en el escritorio	Unir / Snap
Envoltura	forma de bloque de comandos que empieza y termina con el mismo comando inicial de control	
Tile	cada una de las celdas gráficas de 32x32 pixels de los mapas	Celda / Casilla

- Acrónimos utilizados en el marco del desarrollo iterativo y el Proceso Unificado

Cuadro 2.3: Acrónimos del Proceso Unificado

Término	Descripción
UC	Caso de Uso
UCR	Realización de Caso de Uso
DSS	Diagrama de Secuencia del Sistema
DI	Diagrama de Interacción
DC	Diagrama de Clases
UP	Proceso Unificado

3 Diseño

3.1. Lenguaje de Comandos

La aplicación utiliza un lenguaje de programación gráfico compuesto por los comandos gráficos disponibles en la Paleta de Comandos. Para desarrollar este lenguaje de programación ha sido necesario definir un léxico o tokens del lenguaje, una sintaxis o gramática del lenguaje y unas reglas semánticas que definen las acciones del robot asociadas a cada comando gráfico.

3.1.1. Léxico (Tokens)

Los tokens representan los símbolos del lenguaje de programación, los cuales en nuestro caso, están representados por los comandos gráficos disponibles en la Paleta de Comandos. Internamente tienen asociados comandos lógicos, donde se definen las propiedades de cada uno.

Se pueden clasificar en 3 grandes grupos:

- Control de Flujo
 - Start
 - Repeat
 - Wait
 - If
- Sensores (condiciones C)
 - Color
 - formato (R,G,B)
 - Key
 - carácter {a, b, ... A, B ... Z}
- Actuadores
 - SetPosition
 - setPosition(x,y) - donde x e y son las coordenadas 2D del escenario
 - SetSpeed
 - en un límite de [1 .. 10]

- Move (en principio en las 4 direcciones básicas del escenario)
 - Move up
 - Move down
 - Move right
 - Move left

3.1.2. Sintaxis (Gramática)

La gramática definida para el lenguaje de programación con comandos, en notación BNF, es como sigue:

- Programa := *Start* Bloque;
- Bloque := Sent Bloque | Sent;
- Sent := Sent_if | Sent_repeat | Sent_wait | Actuador;
- Sent_if := *if* Sensor Bloque *endIf*;
 - *endIf* representa otro comando asociado con el mismo IF del inicio
- Sent_repeat := *Repeat* Sensor Bloque *endRepeat*;
 - *endRepeat* representa otro comando asociado con el mismo Repeat del inicio
- Sent_wait := *Wait* Sensor;
- Sensor := *Color* | *Key*;
- Actuador := *Move* | *SetPosition* | *SetSpeed*;

Comentarios:

- En la gramática se indican los símbolos terminales (tokens) con tipografía cursiva y los no-terminales con tipografía normal
- El primer comando válido de un programa debe ser el comando Start
- Como se indica arriba, se ha optado por utilizar dos partes separadas para los bloques de control (repeat e If), de modo que tengan dos comandos asociados , uno de comienzo y otro de fin. Esto nos da cierta libertad a la hora de incluir comandos en un bloque Repeat/ If, permitiendo que un bloque anidado sea tan extenso como queramos, sin embargo nos obliga a utilizar un comando de fin de bloque (end_repeat o end_if) asociado con el comando de inicio, para que la construcción sintáctica sea correcta.
 - dentro de la lógica de la aplicación se ha implementado la gestión del anidamiento haciendo que cada vez que se coloque un nuevo comando repeat o if, se incremente el nivel de anidamiento en 1, decrementándose igualmente en 1 por cada comando end_repeat o end_if que se añada

- Gráficamente, los comandos se acoplan según las reglas del UC2 - Unir dos comandos, de forma que todos se acoplan por abajo, excepto los sensores que se acoplan por la derecha de los de control (repeat, wait, if)

3.1.3. Semántica (Acciones)

A continuación se detalla las acciones semánticas de cada comando del lenguaje.

- Comandos de Control:
 - Start: comienzo de programa.
 - es una etiqueta sintáctica que marca el inicio del programa, pero no tiene significado asociado (en principio ...)
 - Repeat: bucle estilo “repeat S until C”.
 - `while(not C) {
 bloque_sentencias;
}`
 - Wait: espera que se produzca una condición (que uno de los sensores tome el valor esperado).
 - `if(not C) {
 wait();
}`
 - If: si se produce una condición C (sensor con valor determinado) se ejecutan las sentencias indicadas
 - `if(C) {
 bloque_sentencias;
}`
- Sensores:
 - Color: comprueba si el color de la posición del escenario donde se encuentra el Robot coincide con su atributo de color.
 - `this.color == colorRobot`
 - Key: comprueba si se ha pulsado una determinada tecla.
 - `this.key == lastKey`
- Actuadores:
 - SetPosition: determina coordenadas X e Y donde posicionar el robot.
 - `setPosition(int x, int y){
 robot.x = x;
 robot.y = y;
}`

- SetSpeed: determina la velocidad de desplazamiento del robot.

```
○ setSpeed( int s){
    robot.speed = s;
}
```

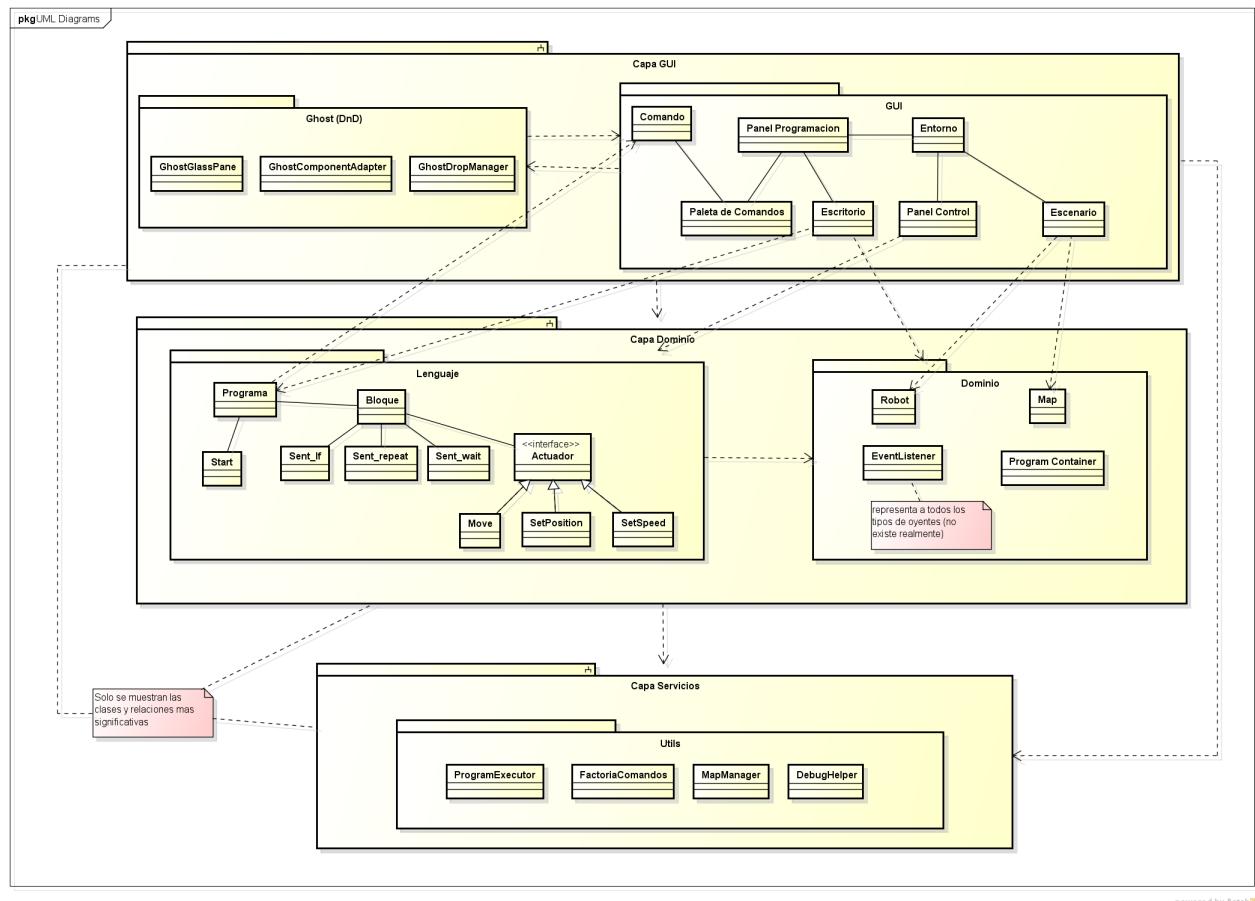
- Move: desplaza el robot un paso en la dirección de movimiento indicada.

```
○ move( int direction ) {
    robot.x += dx;
    robot.y += dy;
}
```

3.2. Arquitectura SW

- Para el diseño de la arquitectura se han tratado de seguir los principios básicos del diseño de objetos, como son el bajo acoplamiento y la alta cohesión, tratando que cada módulo o paquete tenga una serie de clases cohesivas entre si, con responsabilidades definidas y similares, y con bajo acoplamiento entre módulos, el cual cuando se da tiene lugar en interfaces bien definidas.
- En concreto se ha seguido el patrón de arquitectura de Capas, mediante el cual se organiza la estructura lógica a gran escala de la aplicación como un conjunto de capas separadas con responsabilidades distintas entre capas y similares dentro de ellas.
- Así mismo se ha tratado de seguir el principio de diseño de separación Modelo-Vista, de modo que en este caso existe una capa para la gestión de los eventos y responsabilidades de la interfaz gráfica, la cual está separada (y hace uso) de la capa de la lógica del dominio.
 - aunque en principio las capas inferiores no conocen los detalles de las capas superiores, existe al menos una excepción a esto, en el ámbito del manejo del comportamiento del robot. Esto es así debido a que los comandos lógicos (move, setPosition, etc.) acceden al Entorno para llegar al Escenario, que eventualmente mueve (o cambia el estado en general) del Robot, ya que en este caso el Escenario es el Experto en Información de este objeto y es el que le gestiona. Para paliar esta violación de acceso ascendente, se ha utilizado el patrón Singleton para acceder a la única instancia del Entorno, proporcionando un único punto de acceso global, minimizando los perjuicios del acoplamiento.
- Siguiendo estos principios y teniendo en cuenta el alcance de la aplicación se ha optado por una arquitectura de 3 capas, GUI, Dominio y Servicios, la cual se muestra en el siguiente diagrama UML, indicando su estructuración en paquetes y señalando las clases y dependencias más relevantes entre ellas

Figura 3.1: Arquitectura SW - UML



3.3. Diagramas de Clases

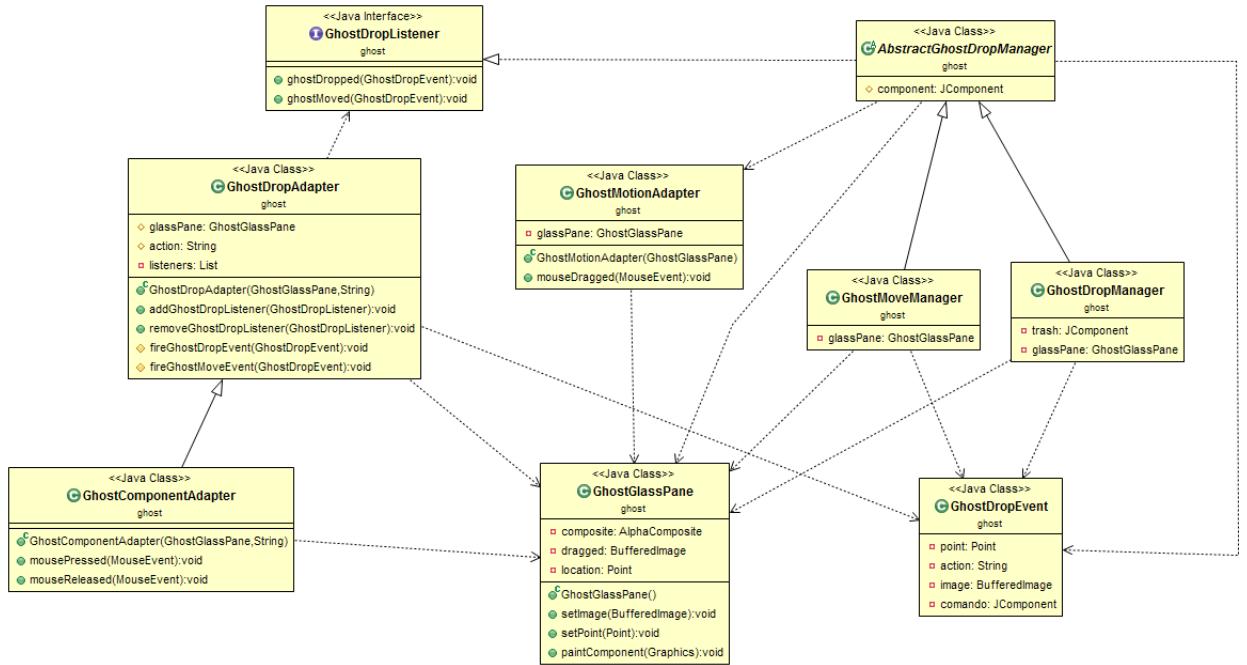
A continuación se muestran los Diagramas de Clases de Diseño mas relevantes de la aplicación, divididos en 3 paquetes, Ghost, Gui y Lenguaje:

3.3.1. DCD Ghost

El paquete Ghost es el encargado de la gestión del drag & drop de los comandos. Su funcionamiento a grandes rasgos se basa en la utilización del GlassPane de javax.swing, que es un panel invisible que forma parte del JRootPane y se coloca delante del resto de paneles siendo muy útil para interceptar eventos de entrada para el root pane.

El uso de este panel junto con una serie de oyentes y gestores de eventos (mostrados en el siguiente diagrama de clases) permite que al hacer click sobre los comandos, se pinten copias sobre el glassPane las cuales se mueven con este panel al arrastrarlo por el entorno, y finalmente al soltar el click se depositan sobre el Escritorio, creando una nueva copia del comando sobre este nuevo panel, y borrando el comando del glassPane.

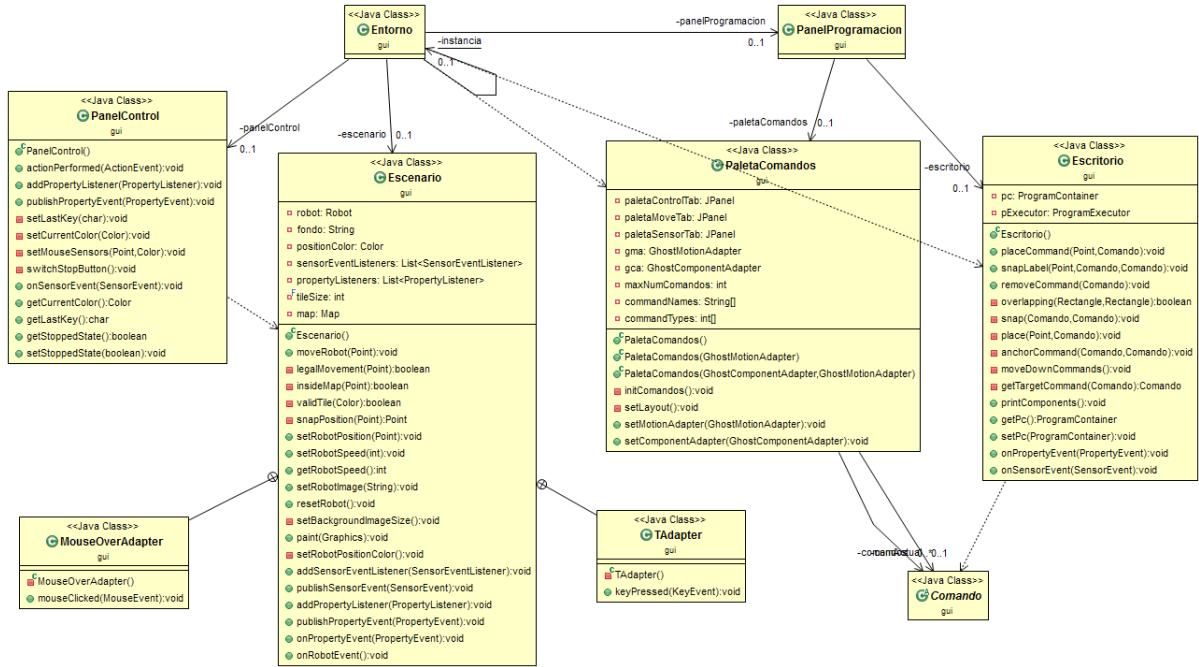
Figura 3.2: DCD paquete Ghost



3.3.2. DCD GUI

Este paquete es el encargado de la gestión gráfica de la aplicación (junto con el Ghost). Se compone como se puede apreciar, de los diferentes paneles del entorno gráfico junto con una clase maestra para el entorno en si, la cual es la clase principal de la aplicación, conteniendo al resto de paneles y gestionando la inicialización y configuración del entorno.

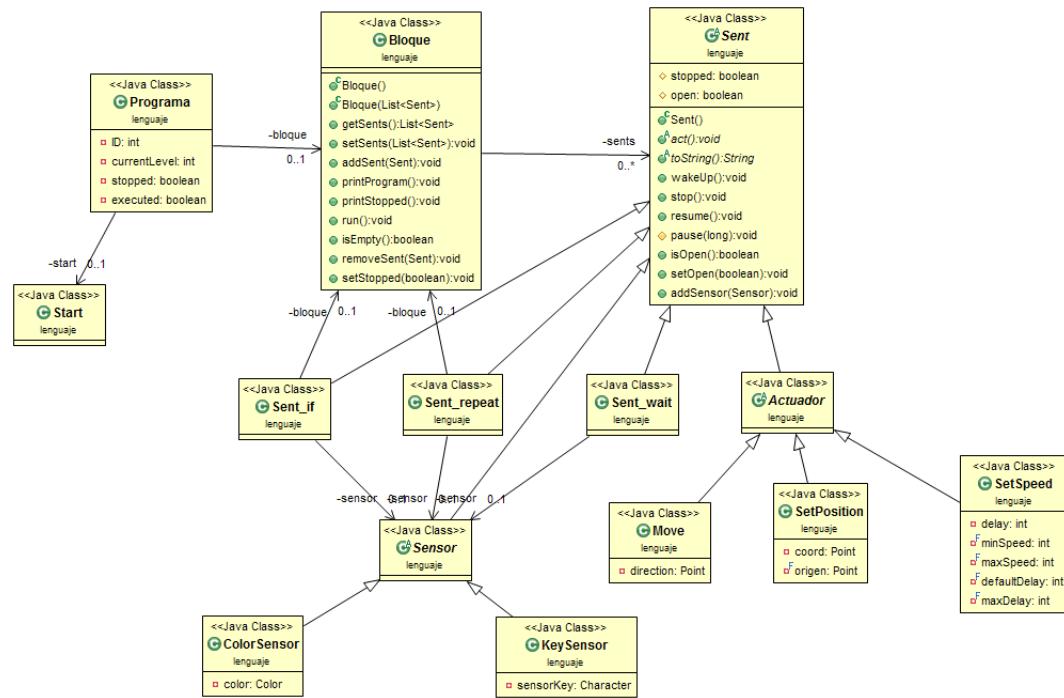
Figura 3.3: DCD paquete GUI



3.3.3. DCD Lenguaje

Se incluye en este paquete toda la lógica de gestión del lenguaje de programación creado. Como se puede ver, la organización del paquete sigue la misma estructura que la gramática del lenguaje, ya que de esta forma se define la estructura sintáctica de un programa. Por su parte dentro de cada clase se encuentran definidos los atributos y métodos pertinentes que permiten implementar la semántica del lenguaje.

Figura 3.4: DCD paquete lenguaje

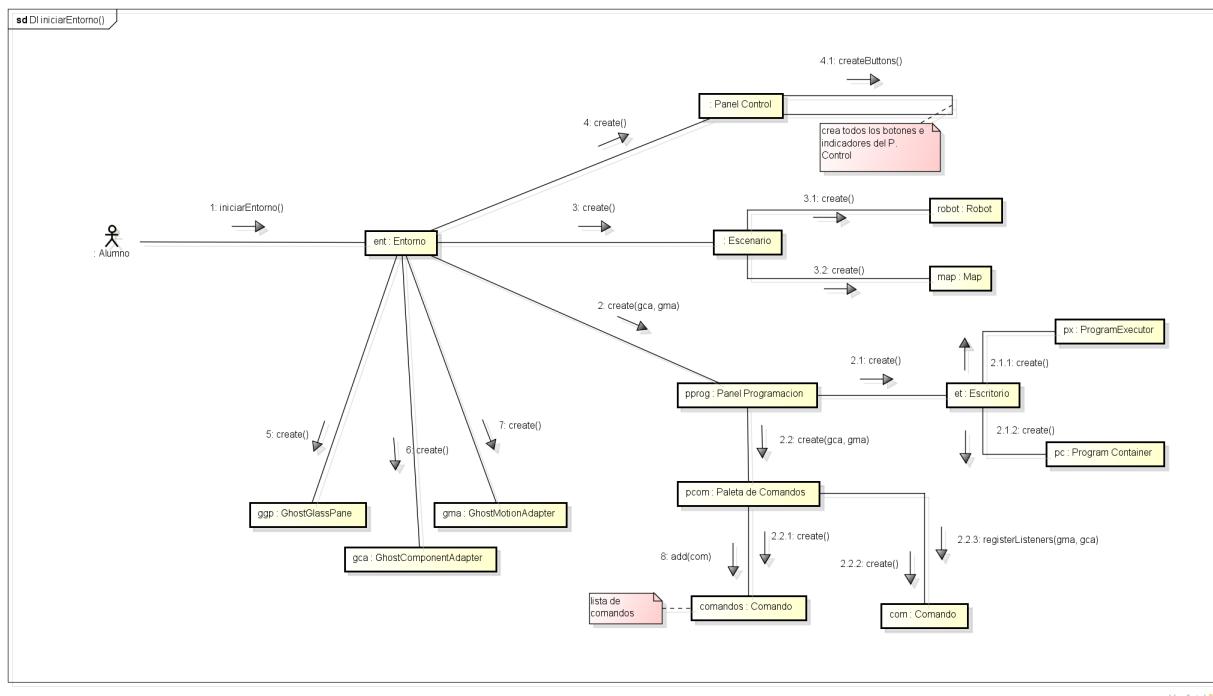


3.4. Diagramas de Interacción

A continuación se muestran los principales diagramas de interacción creados en el proceso de desarrollo, los cuales representan la dinámica del sistema en las principales operaciones definidas a consecuencia del análisis de requisitos, identificadas en el Modelo de Casos de Uso y adicionalmente en los contratos de las operaciones.

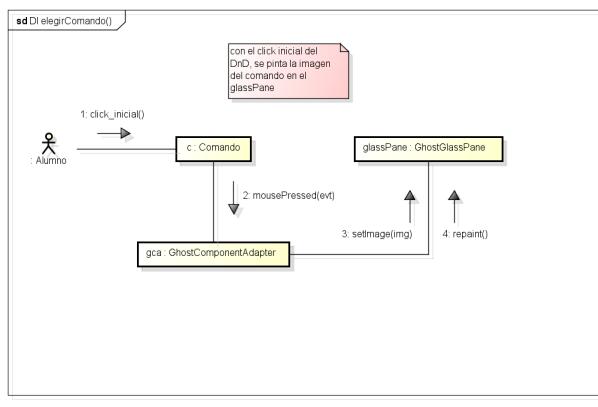
DI iniciarEntorno()

Figura 3.5: DI iniciarEntorno()



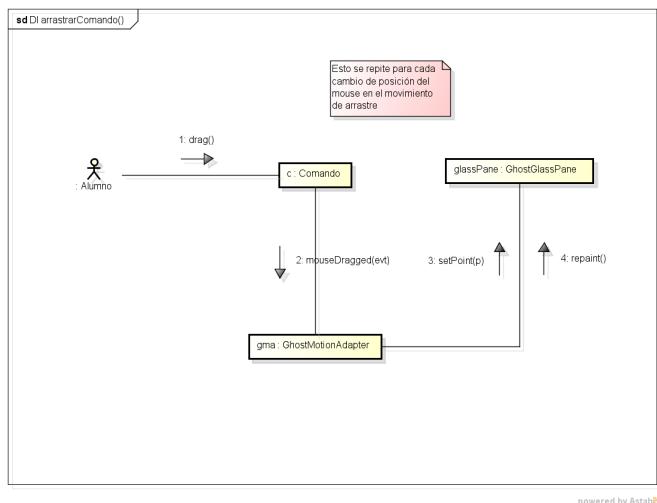
DI elegirComando()

Figura 3.6: DI elegirComando()



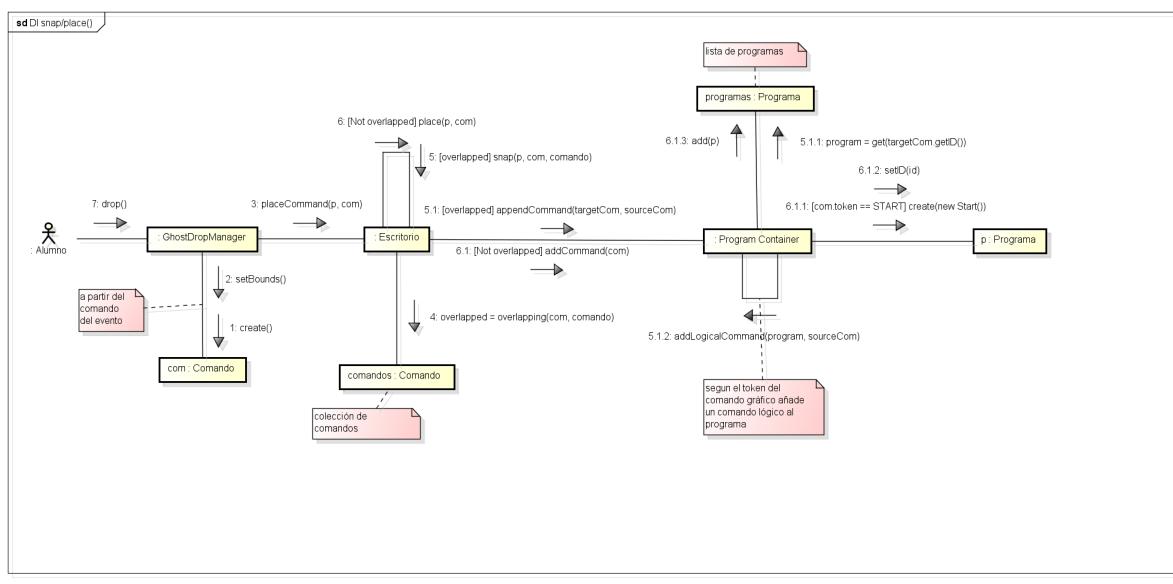
DI arrastrarComando()

Figura 3.7: DI arrastrarComando()



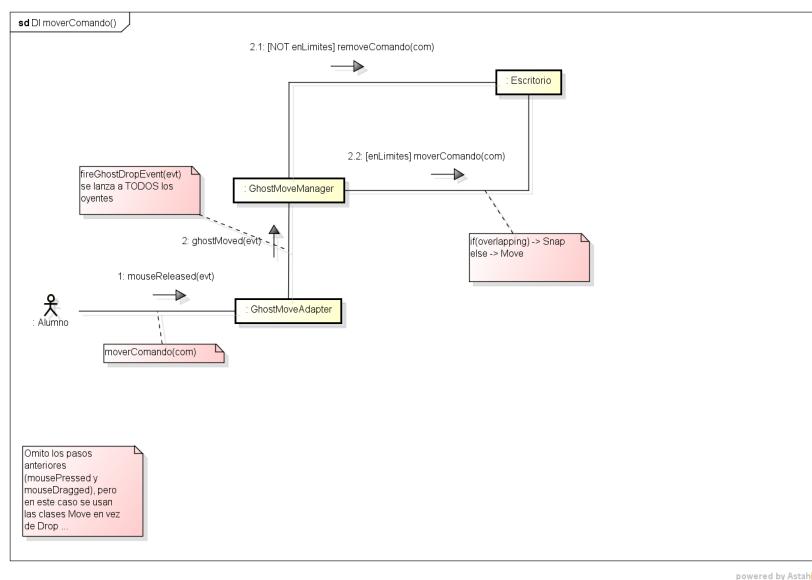
DI snap/place()

Figura 3.8: DI snap/place()



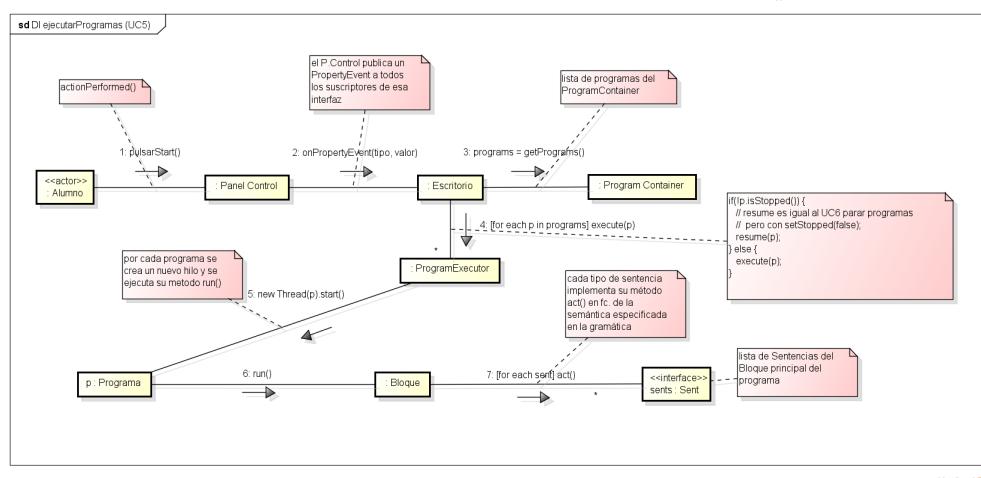
DI moverComando()

Figura 3.9: DI moverComando()



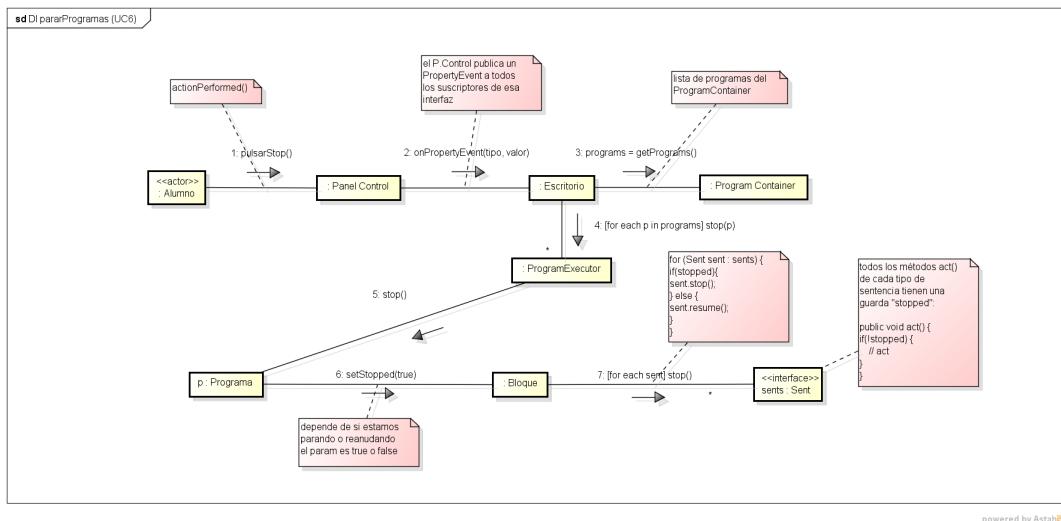
DI ejecutarProgramas()

Figura 3.10: DI ejecutarProgramas()



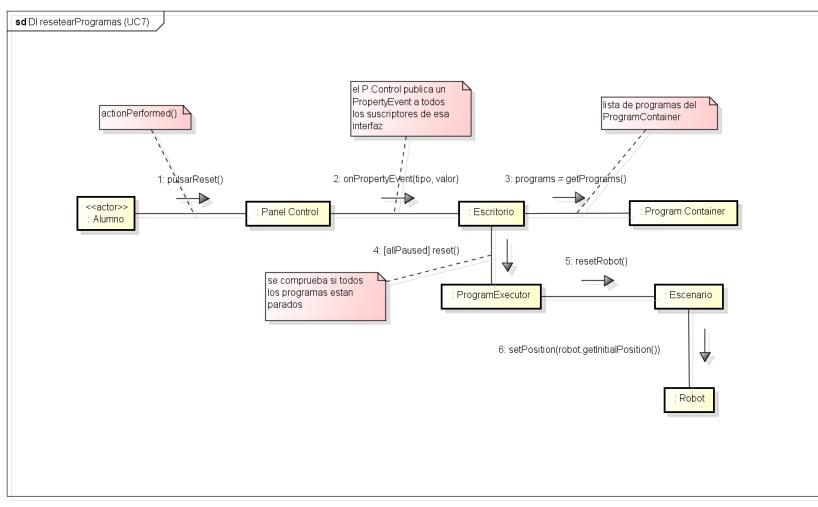
DI detenerProgramas()

Figura 3.11: DI detenerProgramas()



DI resetProgramas()

Figura 3.12: DI resetProgramas()



4 Implementación

4.1. Tecnologías Utilizadas

Para la implementación del proyecto se han utilizado las siguientes tecnologías y lenguajes:

- Lenguaje de desarrollo: Java
 - jdk 1.7.0_25
 - jre7 (Version 7 Update 55)
 - las razones por las que se opta por este lenguaje son a grandes rasgos las siguientes:
 - es orientado a objetos y por tanto adecuado para la representación de las entidades del problema identificadas en el análisis preliminar. Aparte de esto la posibilidad de reutilización de código es otro punto a tener en cuenta para la elección de este lenguaje
 - es independiente de la plataforma y por tanto posibilita la portabilidad de la aplicación a otros dispositivos y SO's.
 - su soporte de hilos le hace atractivo para la implementación de varios programas corriendo a la vez en la aplicación
 - el conocimiento previo acumulado en este lenguaje de programación (en especial en el desarrollo del compilador de HAda en el curso 3º)
- IDE's:
 - Eclipse (v. Juno Service Release 1) como entorno principal de desarrollo
 - NetBeans (v. 7.3.1) como entorno complementario para diseño de elementos y layout de la GUI
- SO:
 - Windows 7 Home Premium SP1
- Software adicional:
 - Astah Community 6.8.0 como software de diseño UML
 - LyX 2.0.1 como editor de texto
 - Illust Studio v.1.2.7 como software de edición de imagen

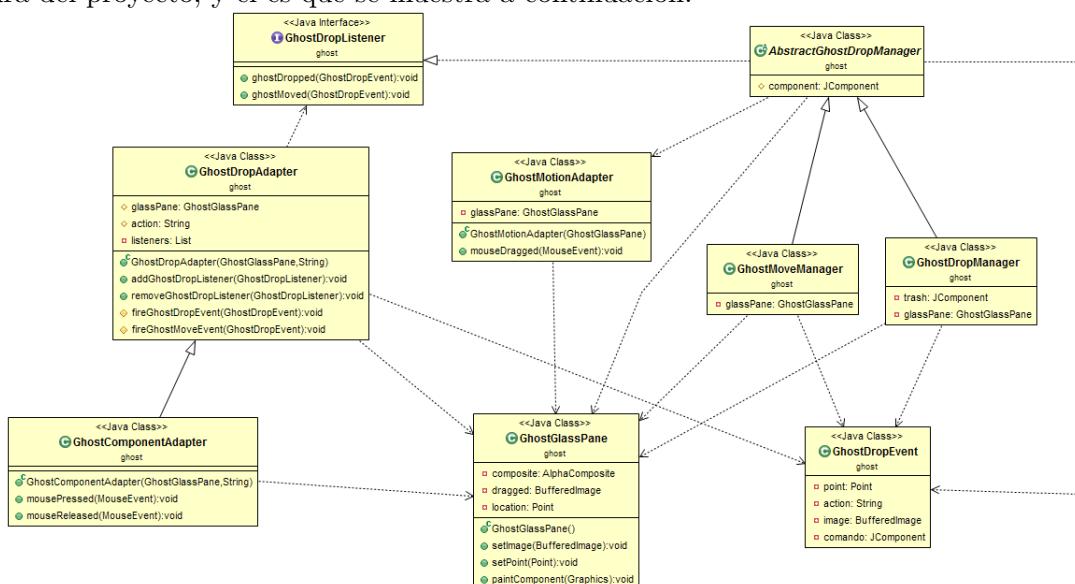
- Photoshop CS5 v.12.0 (white rabbit release) como software de edición de imagen
- JSsmooth para la conversión a .exe del ejecutable en .jar

4.2. Aspectos Especiales de Implementación

4.2.1. Lógica del Drag and Drop

Un aspecto especialmente relevante en el proyecto es la mecánica de las acciones del arrastre y colocación de comandos en el entorno gráfico. Dicha mecánica está contenida dentro del *paquete ghost*, el cual contiene las clases que se encargan de la implementación de todas las acciones relacionadas con este aspecto.

El diagrama de clases de este paquete se encuentra en el directorio UML de la estructura del proyecto, y el es que se muestra a continuación:

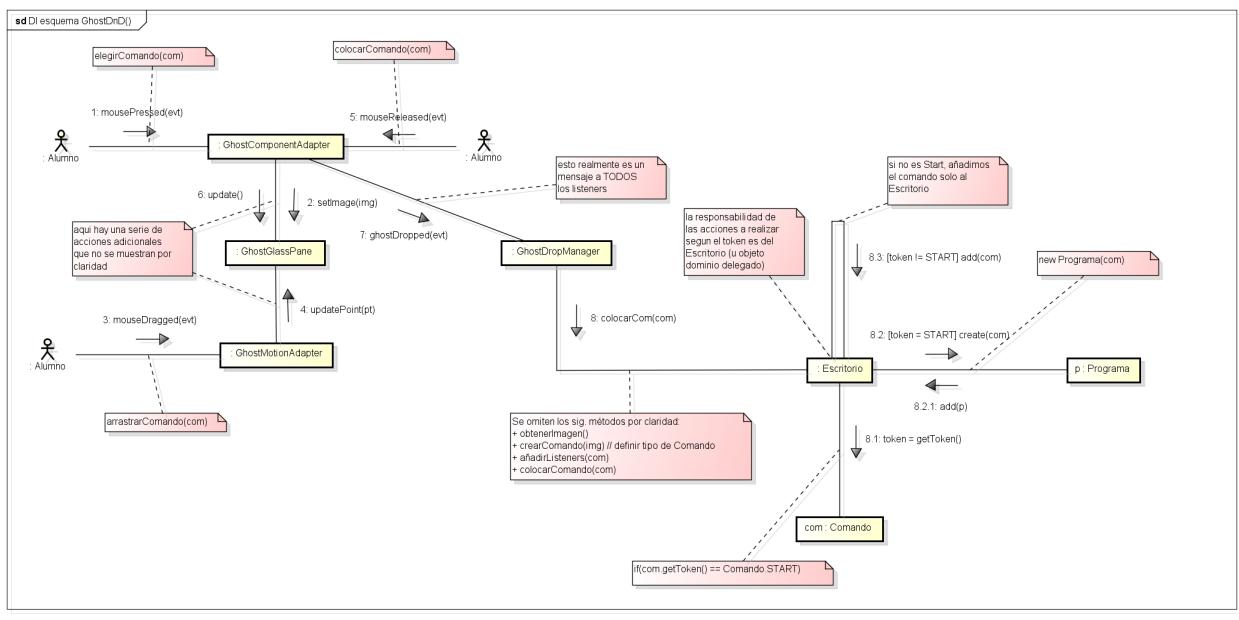


Asimismo, el aspecto dinámico del funcionamiento de este paquete se muestra de forma esquemática en los diagramas de interacción siguientes:

- DI elegirComando()
- DI arrastrarComando()
- DI snap/place()
- DI moverComando()
- Estos diagramas se encuentran también disponibles en el directorio de diseño del proyecto.

En el siguiente diagrama de interacción se muestra la dinámica general del DnD

4.2.1.1. DI GhostDnD()



4.2.2. Lógica de la Creación de Programas

La creación de programas lógicos se inicia cuando se coloca un Comando START en el Escritorio, momento en el cual se crea un nuevo programa, como se puede apreciar en la parte derecha del diagrama DI GhostDnD().

La creación lógica se encapsula en el método `addComando()` del atributo `ProgramContainer` del Escritorio, el cual se muestra a continuación:

```
/**
 * crea un programa si el comando lógico es START
 * @param com el comando gráfico cuyo comando lógico debe ser START para crear un nuevo programa
 */
public void addComando(Comando com) {
    if (com.getToken() == Comando.START) {
        //crear nuevo programa con un start
        Programa p = new Programa(new Start());
        // añadimos la ID del siguiente indice de la lista de programas {0 .. size}
        int id = getNextId();
        com.setID(id);
        p.setID(id);
        programs.add(p);
        DebugHelper.debugMessage("New Program created: ID = " + p.getID());
    }
}
```

Una vez que existe un programa vacío (solo con el Start) en el Escritorio, se puede seguir construyendo el programa mediante sucesivas acciones de snap válidas, es decir, acoplándole más comandos gráficos, momento en el cual (si el acoplamiento se produce, es decir, si sintácticamente es un programa válido) se añade una nueva sentencia lógica

al programa (encapsulada en el comando lógico del comando gráfico que se acopla). Esta dinámica se puede apreciar en el DI snap/place()

La responsabilidad de esto recae de nuevo en el ProgramContainer, mediante su método *appendComando()* el cual llama a su método privado *addLogicalCommand()*:

```
/*
 * añade un comando lógico al programa cuya ID corresponda con la ID del comando objetivo
 * @param targetCom el comando objetivo cuya ID nos determina el programa donde añadirle
 * @param sourceCom el nuevo comando que queremos añadir al programa
 */
public void appendComando(Comando targetCom, Comando sourceCom) {
    // para añadir un comando (lógico) a un programa tenemos que identificar
    // a que programa pertenece el comando target (gráfico) donde se hace el snap
    // de modo que tiene que existir algún tipo de relación entre comando gráfico y lógico
    // quizás algún tipo de ID de programa
    int programID = targetCom.getID();
    Programa program = programs.get(programID);
    sourceCom.setID(programID);
    // añadimos el comando lógico según el tipo de comando del comando gráfico
    // program.setBloque(new Bloque());
    addLogicalCommand(program, sourceCom, targetCom);
    DebugHelper.debugLine("New Com append to Program (" + programID + ")");
}

/**
 * en este método se encapsula la lógica de creación de programas lógicos ,
 * añadiendo comandos lógicos según los casos ,
 * y construyendo de este modo los programas
 * @param p el Programa actual
 * @param sourceCom el comando origen que queremos añadir
 * @param targetCom el comando destino del acoplamiento , usado solo para los casos Sensores
 */
private void addLogicalCommand(Programa p, Comando sourceCom, Comando targetCom){
    Sent sent = null;
    switch (sourceCom.getToken()) {
        case Comando.SETPOSITION:
            sent = ((ComandoPosition) sourceCom).getLogicalCom();
            p.addSent(sent);
            break;
        case Comando.SETSPEED:
            sent = ((ComandoSpeed) sourceCom).getLogicalCom();
            p.addSent(sent);
            break;
        case Comando.MOVE:
            sent = ((ComandoMove) sourceCom).getLogicalCom();
            p.addSent(sent);
            break;
        case Comando.COLOR:
            Sensor sensorColor = (Sensor) ((ComandoColor) sourceCom).getLogicalCom();
            Sent targetSent = targetCom.getLogicalCom();
            // por las restricciones del acoplamiento del canSnap()
            // el targetSent solo debería ser del tipo Sent_repeat / Sent_if / Sent_wait
            // internamente la clase Sent se encarga de verificar a que clase pertenece
            // y si su sensor es nulo (addSensor())
            targetSent.addSensor(sensorColor);
            // p.addSensor(sensorColor);
            // sent = new ColorSensor(c.getRed(),c.getGreen(),c.getBlue());
            break;
        case Comando.KEY:
            Sensor sensorKey = (Sensor) ((ComandoKey) sourceCom).getLogicalCom();
            Sent targetSent1 = targetCom.getLogicalCom();
    }
}
```

```

        // por las restricciones del acoplamiento del canSnap()
        // el targetSent solo debería ser del tipo Sent_repeat / Sent_if / Sent_wait
        // internamente la clase Sent se encarga de verificar a que clase pertenece
        // y si su sensor es nulo (addSensor())
        targetSent1.addSensor(sensorKey);
        // p.addSensor(sensorKey);
        //sent = new Key();
        break;
    case Comando.REPEAT:
        sent = ((ComandoRepeat) sourceCom).getLogicalCom();
        // la marcamos como sentencia con bloque abierto
        sent.setOpen(true);
        p.addSent(sent);
        // incrementamos el nivel de anidamiento
        p.setLevel(p.getLevel() + 1);
        //System.out.println("nivel de anidamiento: " + p.getLevel());
        break;
    case Comando.END_REPEAT:
        // obtenemos la ultima sentencia con bloque abierto y la cerramos
        p.closeCurrentBlock();
        // decrementamos el nivel de anidamiento
        p.setLevel(p.getLevel() - 1);
        //System.out.println("nivel de anidamiento: " + p.getLevel());
        break;
    case Comando.IF:
        sent = ((Comandolf) sourceCom).getLogicalCom();
        // la marcamos como sentencia con bloque abierto
        sent.setOpen(true);
        p.addSent(sent);
        // incrementamos el nivel de anidamiento
        p.setLevel(p.getLevel() + 1);
        //System.out.println("nivel de anidamiento: " + p.getLevel());
        break;
    case Comando.END_IF:
        // obtenemos la ultima sentencia con bloque abierto y la cerramos
        p.closeCurrentBlock();
        // decrementamos el nivel de anidamiento
        p.setLevel(p.getLevel() - 1);
        //System.out.println("nivel de anidamiento: " + p.getLevel());
        break;
    case Comando.WAIT:
        sent = ((ComandoWait) sourceCom).getLogicalCom();
        p.addSent(sent);
        break;
    default:
        // no pueden ocurrir mas casos en principio
        break;
    }
    //p.addSent(sent);
}

```

4.2.3. Lógica de la Ejecución de Programas

4.2.3.1. Start

- Para la ejecución de los programas se ha optado por delegar dichas acciones en un gestor de hilos sencillo, el ProgramExecutor, que se encarga de lanzar un nuevo hilo por cada programa presente en el Escritorio, una vez que se recibe el evento de pulsación del botón Start del Panel de Control, de forma que existirá un hilo

principal y un hilo por cada programa en ejecución.

- Una vez que se crea el hilo para el programa en cuestión, se invoca implícitamente a su método run(), el cual llama al método recursivo run() de su bloque de sentencias, ejecutando secuencial y recursivamente (para las sentencias con bloques anidados, como Repeat e If) las acciones semánticas de cada sentencia, y consecuentemente ejecutando las acciones programadas del Robot.
- Esta dinámica se puede observar en el DI ejecutarProgramas()

4.2.3.2. Stop

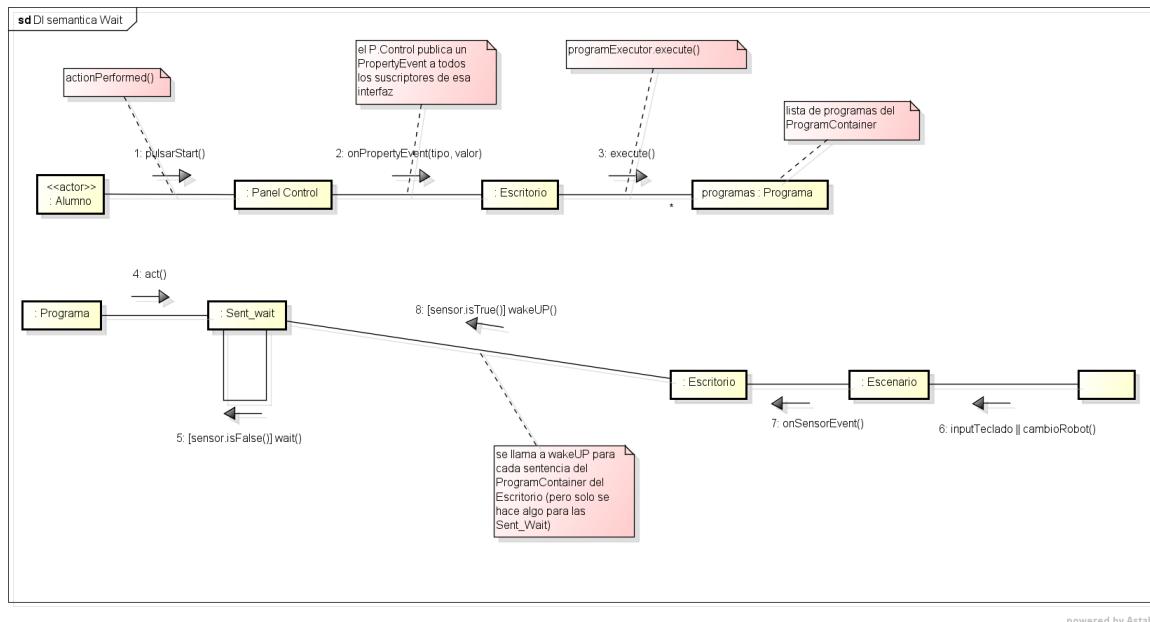
- Los programas ejecutados siguen su curso hasta que finalizan, finalizando la ejecución del hilo, o bien hasta que son detenidos, bien por una pulsación del botón Stop del Panel de Control, o porque el Robot ha realizado un movimiento ilegal (fuera de límites en Mapas Abiertos o fuera del camino en Mapas Cerrados)
- La lógica de la parada de programas se implementa mediante un atributo booleano en los programas y en cada sentencia, stopped, el cual permite o prohíbe la ejecución de las acciones semánticas de cada sentencia
- El diagrama de interacción DI detenerProgramas() muestra dicha dinámica de forma gráfica

4.2.4. Semántica de la sentencia Wait

- Sent _Wait: se ha realizado una implementación de la semántica del wait, usando los métodos wait() y notify() de java.Thread, reduciendo de este modo el consumo de ciclos de CPU que se daba con la implementación de la espera ocupada anterior.
 - al pulsarse el botón Start del P.Control se ejecutan todos los programas del Escritorio. Para esto se llama a la única instancia (singleton) del ProgramExecutor (encargado de la gestión de concurrencia) que se encarga de ejecutar cada Programa lógico, creando un hilo diferente para cada uno
 - si un Programa lógico contiene una o más sentencias Wait, se llama al método wait() de esa sentencia, lo cual pone en espera el hilo que la contiene (el hilo de ese programa)
 - para despertar al hilo, cada vez que el Escenario recibe un input de teclado o cambia la posición del Robot (y por tanto el sensor *color actual*), envía un eventoSensor a todos los objetos que implementen la interfaz SensorEventListener, como es el caso del Escritorio
 - al recibir el eventoSensor, el Escritorio llama a un método wakeUP en cada sentencia de cada programa, los cuales son vacíos excepto en el caso de las sentencias If y Repeat, que llaman recursivamente a los wakeUP de sus propios bloques de sentencias, y en el caso de la propia sentencia Wait, en el cual se

comprueba si coincide su sensor con el del P.Control, y en caso afirmativo se llama al notify() del hilo de ese programa

Figura 4.1: DI wait()



powered by Astah

4.3. Arquitectura - Paquetes

4.3.1. Dominio

- En este paquete se encapsulan todas las clases que pertenecen al dominio lógico de la aplicación, pero que no tienen que ver con la gramática del lenguaje de programación diseñado

4.3.2. Ghost

- Incluye todas las clases encargadas de la gestión de las acciones del Drag and Drop

4.3.3. GUI

- Agrupa a todas las clases gráficas que aparecen en la GUI de la aplicación

4.3.4. Lenguaje

- Se incluyen aquí todas las clases del dominio relacionadas con la gramática del lenguaje de programación diseñado

4.3.5. Utils

- Clases utilitarias para realizar diversas funciones de apoyo y servicio a la aplicación

5 Pruebas

En este epígrafe se incluye el plan de pruebas realizado, desglosado según las principales áreas de prueba.

Los comentarios, notas y aclaraciones se mantienen por su valor documental para su posterior uso en futuras versiones de la aplicación

5.1. Comandos

5.1.1. Snap

5.1.1.1. Start:

- Colocación en Escritorio solo:
 - debería crearse un nuevo programa con un ID nuevo y secuencial desde 0 — OK
- Snap a otros comandos:
 - no debería ocurrir ya que Start no puede acoplarse a ninguno — OK
 - debería quedarse colocado (placed) en el Escritorio y formarse otro nuevo programa con el siguiente ID — OK
 - esto plantea problemas de visibilidad ya que los nuevos se colocan detrás de los antiguos, de modo que puede ocultarse parcial / totalmente
- Movimiento desde el Escritorio:
 - los Comandos Start (o Inicio) una vez colocados en el Escritorio no pueden moverse (locked == true)
 - esto plantea problemas cuando quedan ocultos por otros (no pueden recolocarse en caso de que sean parcialmente visibles)

5.1.1.2. Repeat:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK

- debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — *OK*
- no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — *OK*
- Cierre de Bloque:
 - debería tener asociado un comando End_Repeat que cierre el bloque repetir (disminuyendo el nivel de anidamiento) — *OK*
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — *OK*
 - una vez acoplado no debería poder moverse — *OK*

5.1.1.3. Wait:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — *OK*
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — *OK*
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — *OK*
 - no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — *OK*
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — *OK*
 - una vez acoplado no debería poder moverse — *OK*

5.1.1.4. If:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — *OK*
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — *OK*
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — *OK*

- no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.5. End_Repeat:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
 - no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.6. End_If:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
 - no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.7. Move:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
 - no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.8. Set Position:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
 - no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.9. Set Speed:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:

- se debería poder acoplar a todos excepto a los Sensores (Color y Key), siempre que estén libres (downSlot == true) — OK
- debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
- no debería acoplarse a otro comando si está ocupado el hueco correspondiente (downSlot en este caso) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.10. Color:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar solo a los comandos Repeat, Wait e If, siempre que estén libres (rightSlot == true) — OK
 - en ocasiones no se acopla a uno de los candidatos aunque estén libres (*pendiente de ver por que*)
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK
- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.11. Key:

- Colocación en Escritorio solo:
 - no debería crearse ningún programa lógico — OK
- Snap a otros comandos:
 - se debería poder acoplar solo a los comandos Repeat, Wait e If, siempre que estén libres (rightSlot == true) — OK
 - en ocasiones no se acopla a uno de los candidatos aunque estén libres (*pendiente de ver por que*)
 - debería quedarse acoplado (snapped) al programa lógico correspondiente y tener su mismo ID (acreditando que forma parte de este) — OK

- Movimiento en Escritorio:
 - debería poder moverse (not locked) — OK
 - una vez acoplado no debería poder moverse — OK

5.1.1.12. GENERAL:

- los comandos se acoplan en función de unas posiciones fijas, que funcionan para el tamaño de las imágenes actuales, pero que al modificarse pueden no ser válidos
- ```
private void anchorCommand(Comando targetCom, Comando sourceCom) {
 int targetX = targetCom.getBounds().x;
 int targetY = targetCom.getBounds().y;
 int targetHeight = targetCom.getBounds().height;
 int targetWidth = targetCom.getBounds().width;

 if(sourceCom.getToken() == Comando.COLOR || sourceCom.getToken() == Comando.KEY) {
 sourceCom.setBounds(targetX + targetWidth - targetWidth / 7, targetY + 7, targetWidth, targetHeight);
 } else {
 sourceCom.setBounds(targetX, targetY + targetHeight - targetHeight / 2, targetWidth, targetHeight);
 }
}
```

#### 5.1.2. JComboBox / JFormattedTextField

- El comboBox se crea solo para los comandos necesarios (todos excepto los de tipo control) — OK
- El JComboBox se ubica apropiadamente en la imagen de los comandos
  - se está realizando la ubicación manualmente, por prueba y error
  - quizás sea necesario rediseñar algunos comandos para acomodar bien la caja del comboBox o textField

#### 5.1.2.1. Comando Move

- se eligen imágenes cargadas desde el directorio del proyecto (.png) para definir las flechas de movimiento (up, right, down, left)
- las imágenes se cargan correctamente — OK
  - pendiente de ver como se comporta cuando exportemos la aplicación al ejecutable .jar — OK
    - necesita el subdirectorio Imagenes en el mismo directorio de instalación en que esté el .jar
- se puede seleccionar cualquiera de las 4 direcciones básicas — OK
- cada dirección cambia realmente la dirección de movimiento del Robot — OK
- pendiente de asociar cada dirección de movimiento con un .gif diferente del Robot (mirando en la dirección adecuada)

### 5.1.2.2. Comando Set Position

- se usan dos cajas de texto que aceptan enteros, para definir las posiciones en X e Y del Robot dentro del Escenario
- las imágenes se cargan correctamente — OK
- las cajas de texto solo aceptan enteros — OK
  - al introducir caracteres o números no enteros, los rechaza y vuelve a mostrar 0
- al introducir valores fuera de límites, los rechaza, vuelve al valor por defecto y muestra dicho valor (PTE)
- cada posición cambia realmente la posición del Robot dentro del Escenario — OK
- si se introducen unas coordenadas fuera de los límites del Escenario no ocurre nada — OK
  - posibilidad de informar con un mensaje de diálogo de esto
- en Mapas Cerrados, si se introducen unas coordenadas fuera del camino no ocurre nada — OK
  - posibilidad de informar con un msg de diálogo
- en Mapas Cerrados, independientemente de la posición que se introduzca, la posición visual se ajusta al centro del tile válido correspondiente (`snapPosition()`) — OK
- en Mapas Abiertos, quizás se debería ajustar al centro del tile

### 5.1.2.3. Comando Set Speed

- se usa una caja de texto que acepta enteros, para definir la velocidad del Robot (la cual determina el retraso de las acciones — perceptible en bucles solamente)
- las cajas de texto solo aceptan enteros — OK
  - al introducir caracteres o números no enteros, los rechaza y vuelve a mostrar 0
- al introducir valores fuera de límites, los rechaza, vuelve al valor por defecto y muestra dicho valor (PTE)
- el valor introducido en la caja de texto cambia realmente la velocidad de ejecución de las acciones (el delay de los bucles) — OK

#### 5.1.2.4. Comando Sensor Color

- para definir los colores del sensor ComandoColor se usa la lista desplegable del JComboBox, la cual contiene una serie de Colores predefinidos, así como un JColorChooser al que se accede haciendo clic en la caja del JComboBox, el cual permite seleccionar cualquier Color disponible en Java.
- al seleccionar un Color vía ColorChooser o lista desplegable, se mantiene la selección de ese color visible en la caja del comando gráfico — OK
- al seleccionar un Color del ColorChooser se cambia realmente el color del sensor del comando lógico asociado a ese comando gráfico — OK (visto por salida del debug)
- al seleccionar un Color con la lista desplegable se cambia realmente el color del sensor del comando lógico asociado a ese comando gráfico — OK

#### 5.1.2.5. Comando Sensor Key

- se utiliza la lista desplegable del JComboBox desde la cual se puede elegir cualquier carácter del alfabeto inglés en el rango [a-z] (minúsculas, no incluida la ñ, no incluye números).
- al seleccionar cualquier carácter de la lista desplegable se establece realmente dicho carácter como sensor del comando lógico asociado al comando gráfico — OK
- al introducir manualmente cualquier carácter en la caja de texto se establece realmente dicho carácter como sensor del comando lógico asociado al comando gráfico
  - no existe la posibilidad de definirlos manualmente - (lanza «java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Character»)

#### 5.1.2.6. GENERAL

- las cajas de los comandos se ubican en función de unas posiciones fijas, que funcionan para el tamaño de las imágenes actuales, pero que al modificarse pueden no ser válidos

## 5.2. Creación de Programas

- para verificar que los programas se construyan correctamente usamos el método «printProgram()» de la clase Bloque, y cambiamos el Program.run() para que llame a este método
- para la ejecución basta con reemplazar la llamada a «bloque.printProgram()» por «bloque.run()» dentro del método Program.run()

**5.2.1. Start:**

- debe constituir la primera sentencia de un programa lógico — OK
- por si solo constituye el programa vacío
- No debería tener (en principio) una semántica definida, mas allá de permitir la creación de un nuevo programa — OK
  - no tiene un comando lógico asociado (solo sirve como condición de la creación de un nuevo programa lógico)

**5.2.2. Repeat:**

- por si solo no constituye ningun programa — OK
- añade una sentencia Repeat vacía (Sent\_repeat) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- debe aumentar el nivel de anidamiento en 1 cuando se entra en el bloque repeat — OK
- debe estar marcado como «abierto» mientras no se le acople un comando End\_repeat — OK
- debe tener asociado un End\_Repeat, el cual debe disminuir el nivel de anidamiento en 1 y cerrar el bloque (marcarlo como cerrado) — End\_Repeat:
- debe tener asociada una condición — OK
  - los sensores (condiciones) se le pueden agregar en cualquier orden — OK
  - valorar la posibilidad de que un Repeat [null] (un Repeat sin Sensor), equivalga a un bucle infinito (Repeat(true))
- la semántica esta correctamente definida — OK
  - se ha cambiado a un while normal (antes estaba definido como un do-while) para facilitar la comprensión por parte de los alumnos
  - ```
while (!stopped && !sensor.isTrue()) {
    bloque.run();
}
```
 - se ha añadido la guarda !stopped a la condición del bucle, de modo que cuando se paran los programas, los bucles terminan, y los hilos finalizan su ejecución — OK
- añadido el atributo *boolean open* a las Sent, para posibilitar la comprobación de que los bloques de las sentencias de control están abiertos / cerrados, y la correcta construcción de programas con bloques anidados — OK

- solucionado el siguiente problema encontrado durante las pruebas:

```
// Si construimos el programa mostrado a continuación
// no reconoce el sensor del primer If (lo pone a null)
// Además la información de debug nos muestra lo siguiente ,
// indicando que el orden de los Ifs es el inverso

Start;
Repeat(black);
    If(a);
        Move(up);
    End_if;
    If(white);
        SetPosition(0,0);
    End_if;
End_repeat;

#####
# salida #####
#####

-----init program (0)-----
__ init repeat __
Sent_Repeat [Color(R,G,B): (0,0,0)]
__ init if __
Sent_If [Color(R,G,B): (255,255,255)]
Move
SetPosition java.awt.Point[x=0,y=0]
__ end if __
__ init if __
Sent_If [null]
__ end if __
__ end repeat __
-----end program (0)-----


// Ocurre lo mismo cuando encadenamos varias
// sentencias Repeat secuencialmente , pero no cuando están anidadas
// el problema está en el método Programa.getCurrentSent()
// y también en getCurrentBloque() — (están incorrectamente definidos)
```

5.2.3. Wait:

- por si solo no constituye ningun programa — OK
- añade una sentencia Wait vacía (Sent_wait) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- debe tener asociada una condición — OK
- la semántica esta correctamente definida — OK
 - pruebas de la semántica en el Test 6
 - ```
if (!sensor) {
 this.wait();
}
```

- se pueden añadir los sensores de condición de forma desordenada, igual que para el caso del Repeat — OK

- esto es debido al cambio de la mecánica de agregar sensores en el Program-Container, el cual anteriormente iba buscando secuencialmente hasta el primer comando de control, lo cual al ser incorrecto se ha cambiado para que obtenga el comando lógico del comando gráfico objetivo del snap, añadiendo el sensor en dicho comando (wait, repeat o if) si no tiene ya otro sensor asignado

- **solucionado el siguiente problema que aparecía con la anterior mecánica del snap lógico**

- ```
// Al añadir secuencialmente varios Wait seguidos,
// si tratamos de acoplar un comando Sensor a alguno de ellos de forma
// desordenada, el que recibe el sensor lógico es el primero que esté libre,
// de modo que la representación visual no
// corresponde en este caso con el programa lógico creado.

// Esto es debido al uso de la condición
// wait.getSensor() == null en el método Programa.getCurrentSent():
// se busca la primera sentencia wait que esté libre
// y se coloca ahí el sensor, independientemente de que sea ese o no el
// comando gráfico objetivo.

// si tenemos gráficamente el siguiente programa

Start;
Wait(null);
Wait(null);
Wait(null);

// y a continuación acoplamos un comando sensor (gráfico) de esta forma:

Start;
Wait(null);
Wait(black);
Wait(null);

// la salida (programa lógico) que obtenemos es la siguiente:
#####
##### salida #####
#####
-----init program (0)-----
Sent_wait [Color(R,G,B): (0,0,0)]
Sent_wait [null]
Sent_wait [null]
-----end program (0)-----
```

5.2.4. If:

- por si solo no constituye ningun programa — OK
- añade una sentencia If vacía (Sent_if) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- debe aumentar el nivel de anidamiento en 1 cuando se entra en el bloque if — OK

- debe estar marcado como «abierto» mientras no se le acople un comando End_if al bloque del If anterior — OK
- debe tener asociado un End_if, el cual debe disminuir el nivel de anidamiento en 1 y cerrar el bloque (marcarlo como cerrado) — End_if:
- debe tener asociada una condición — OK
 - los sensores (condiciones) se le pueden agregar en cualquier orden — OK
 - valorar la posibilidad de que un If [null] (un If sin Sensor), equivalga a un If(true)
- la semántica esta correctamente definida — OK
 - ```
if(sensor) {
 // hacer acciones del bloque
}
```
- añadido el atributo *boolean open* a las Sent, para posibilitar la comprobación de que los bloques de las sentencias de control están abiertos / cerrados, y la correcta construcción de programas con bloques anidados — OK

#### **5.2.5. End\_Repeat:**

- por si solo no constituye ningun programa — OK
- no tiene un comando lógico asociado, solo sirve como cierre del anterior bloque repeat abierto, y para la gestión del anidamiento — OK
- debe reducir en 1 el nivel de anidamiento — OK
- debe cerrar el anterior bloque repeat abierto — OK
  - NOTA: en realidad dado que su comportamiento es idéntico al end\_if, podría cerrar tanto bloques repeat como bloques if. De este modo se podrían agrupar ambos en un solo comando «End». Posible mejora

#### **5.2.6. End\_If:**

- por si solo no constituye ningun programa — OK
- no tiene un comando lógico asociado, solo sirve como cierre del anterior bloque if abierto, y para la gestión del anidamiento — OK
- debe reducir en 1 el nivel de anidamiento — OK
- debe cerrar el anterior bloque if abierto — OK
  - NOTA: en realidad dado que su comportamiento es idéntico al end\_repeat, podría cerrar tanto bloques repeat como bloques if. De este modo se podrían agrupar ambos en un solo comando «End». Posible mejora

**5.2.7. Move:**

- por si solo no constituye ningun programa — OK
- añade una sentencia Move (Move) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- forma parte del ultimo bloque abierto si existe alguno, o del bloque principal del programa en su defecto — OK
- la semántica esta correctamente definida — OK
  - mueve al Robot una posición en la dirección indicada
  - ```
if (!stopped){
    Entorno.getInstancia().getEscenario().moveRobot(direction);
    pause(Entorno.getInstancia().getEscenario().getRobotSpeed());
}
```
 - se han definido movimientos de 32 pixels, ya que los mapas cerrados usan tiles o casillas de ese tamaño (32x32 px)
 - valorar cambiar esa restricción para mapas abiertos
- se puede cambiar el parámetro dirección asociado en cualquier momento, y se cambiará el atributo real del comando lógico en consecuencia — OK

5.2.8. Set Position:

- por si solo no constituye ningun programa — OK
- añade una sentencia Set Position (SetPosition) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- forma parte del ultimo bloque abierto si existe alguno, o del bloque principal del programa en su defecto — OK
- la semántica esta correctamente definida — OK
 - cambia la posición del Robot en el escenario
 - ```
if (!stopped){
 Entorno.getInstancia().getEscenario().setRobotPosition(coord);
 pause(Entorno.getInstancia().getEscenario().getRobotSpeed());
}
```
- se pueden cambiar los parámetros x e y en cualquier momento, y se cambiarán las coordenadas reales del comando lógico en consecuencia — OK

### 5.2.9. Set Speed:

- por si solo no constituye ningun programa — OK
- añade una sentencia Set Speed (SetSpeed) al último comando colocado en el programa cuya ID corresponda con la ID del comando gráfico objetivo del snap — OK
- forma parte del ultimo bloque abierto si existe alguno, o del bloque principal del programa en su defecto — OK
- la semántica esta correctamente definida — OK
  - en fc. de la velocidad introducida («speed») como parámetro, dentro del rango [1-10], se modifica el delay de las acciones del robot según la fórmula:

$$delay = 1100 - speed * 100$$

```
• if (!stopped){
 Entorno.getInstancia().getEscenario().setRobotSpeed(delay);
}
```

- se puede cambiar su parámetro velocidad en cualquier momento, y se cambiará el atributo real del comando lógico en consecuencia — OK

### 5.2.10. Color:

- por si solo no constituye ningun programa — OK
- añade una sentencia de sensor Color (ColorSensor) a la sentencia de control (Repeat / Wait / If) objetivo del snap (comando lógico asociado al comando gráfico objetivo del snap), siempre que no tenga asignado un sensor de antemano — OK
- la semántica esta correctamente definida — OK
  - comprueba la igualdad entre el color de su atributo y el color actual de la posición del Robot en el Escenario
- se puede cambiar su parámetro color en cualquier momento, y se cambiará el atributo real del comando lógico en consecuencia — OK

### 5.2.11. Key:

- por si solo no constituye ningun programa — OK
- añade una sentencia de sensor tecla (KeySensor) a la sentencia de control (Repeat / Wait / If) objetivo del snap (comando lógico asociado al comando gráfico objetivo del snap), siempre que no tenga asignado un sensor de antemano — OK
- la semántica esta correctamente definida — OK

- comprueba la igualdad entre la tecla de su atributo y el sensor de última tecla presionada registrado en el Panel de Control
- se puede cambiar su parámetro tecla en cualquier momento, y se cambiará el atributo real del comando lógico en consecuencia — OK

## 5.3. Ejecución de Programas

### 5.3.1. Test 1

```
Start;
```

- programa vacío: no realiza ninguna acción — OK

### 5.3.2. Test 2

```
Start;
Move(d); // para d = up, down, right, left;
```

- el Robot se mueve en las 4 direcciones según lo esperado — OK
- al llegar a los bordes del escenario vuelve a la posición inicial (mecánica de los Mapas Abiertos) — OK

### 5.3.3. Test 3

```
Start;
SetPosition(x,y);
```

- el Robot cambia su posición de acuerdo con las coordenadas del programa — OK
- si las coordenadas están fuera de los límites, se muestra un mensaje de diálogo y el Robot permanece en su posición actual — OK

### 5.3.4. Test 4

```
Start;
SetSpeed(s); // probar para diferentes s (dentro y fuera de límites)
Move(d); // para d = up, down, right, left;
```

- el cambio de velocidad no es perceptible en acciones individuales (rango 100-1000 ms), solo dentro de bucles

### 5.3.5. Test 5

```
Start;
If(c) { // probar para sensores de Color y Key
 Move(d);
}
```

- Tecla: el Robot solo se mueve si la ultima tecla presionada, registrada en el sensor del P.Control, coincide con la tecla de la condición del If — OK
- Color: el Robot solo se mueve si el color de su posición actual coincide con el color del sensor del If — OK
  - valorar el uso de algún tipo de imágenes (orbes, señales, globos .... ) para indicar el color de una casilla completa (no solo el pixel que comprueba el robot)

### 5.3.6. Test 6

```
Start;
Wait(c);
Move(d);
```

- Tecla:
  - si la ultima tecla coincide con el sensor del programa el Robot se mueve inmediatamente en la dirección indicada — OK
  - si no coincide el robot no ejecuta ninguna acción y espera hasta que se presiona la tecla del sensor del programa, momento en el cual se mueve a la dirección especificada — OK

### 5.3.7. Test 7

```
Start;
Repeat until (c) {
 Move(d);
}
```

- Tecla:
  - si la ultima tecla coincide con el sensor del programa el Robot no se mueve — OK
    - se ha cambiado la semántica para que se comporte como un *while* normal en vez de un *do-while* — Repeat:
  - si no coincide el sensor, el robot se mueve indefinidamente (hasta el borde del mapa) en la dirección indicada, hasta que se presiona la tecla del sensor del programa, momento en el cual el Robot se detiene — OK
    - en este caso, si se para la ejecución (bien por pulsar el Stop, o porque ha llegado al borde del mapa y se ha parado y reseteado) y se reanuda (con el botón Stop), el Robot NO continua su movimiento (ya que se ha terminado la ejecución de los bucles, y de los hilos de los programas) — OK

- para volver a ejecutar el programa / as, hay que volver a pulsar Start (se crearán nuevos hilos para los programas del Escritorio, ya que los anteriores han finalizado — COMPROBAR), siempre que el Stop esté en la posición correcta (en modo Resume, es decir mostrando el icono de Stop, que es el siguiente estado)
- Color:
  - el Robot se mueve hasta que se topa con una tile donde el color de su posición actual coincide con el sensor del programa — OK

### 5.3.8. Test 8

```
Start;
Repeat until (c) {
 Repeat until (c) {
 Move(d);
 }
}
```

- doble bucle:
  - parece ejecutarse indefinidamente, incluso después de Stop y Clean — OK SOLUCIONADO
  - parece que se ha solucionado: el stop estaba mal implementado, ya que no tenía en cuenta los bloques anidados, de modo que el Stop no afectaba al bucle interno, dando la impresión de que se ejecutaba indefinidamente, incluso después del Stop y Clean (este último no afecta a los programas ya en ejecución; son hilos ya ejecutados)

### 5.3.9. Test 9

```
Start;
SetSpeed(s); // probar con diferentes valores en el rango [1–10]
Repeat until (c) {
 Move(d);
}
```

- en función de la velocidad introducida varía la velocidad del movimiento del Robot — OK
- el movimiento es más rápido cuanto más cercana es la velocidad a 10 y más lento según se acerca a 1 — OK
  - BUG: si el movimiento es lento ( $\approx 1$ ) y el robot choca con el borde del mapa (o las paredes fuera del camino en mapas cerrados), después de volver a su posición original hay un pequeño retardo (igual al delay del robot) hasta que finaliza el programa (y terminan su ejecución los hilos), de forma que si se pulsa Stop (que indicará el paso a modo Resume) en ese intervalo, el programa no termina y el robot vuelve a moverse después de reanudarse la ejecución.

### 5.3.10. Test 10

```

Start;
Repeat until (c) {
 Move(down);
}

Start;
Repeat until (c) {
 Move(right);
}

```

- se inician correctamente, y funcionan como se puede esperar (movimiento diagonal a cada tic del delay), hasta que tocan un borde, momento en el cual se paran todos los programas y se resetea el Robot
  - **bug:** si varios programas usan el Move, y un número par de movimientos son ilegales, se envía un numero par de mensajes SensorEvent.STOP\_SENSOR\_EVENT, los cuales cambian un n° par de veces el estado del botón Stop (la imagen solamente, de momento), dejándolo como al principio)
  - **solución:** ya no se pueden desajustar, ya que ahora al tocar un borde se paran todos los programas en función del estado del Panel de Control (en función del estado del Boton STOP), y si uno toca un borde se lanza un evento de parada + reset, solo si el panel de control no estaba en estado STOPPED, de modo que si llegan dos o mas eventos de movimiento ilegal, solo se lanza UN evento de parada + reset, consiguiendo de este modo un comportamiento coherente del Robot
    - el método switchStopButton() cambia, no solo la imagen del Stop, sino tb su ESTADO (att. *stopped*)

### 5.3.11. Test 11

```

Start;
Repeat until ('a') {
 Move(down);
 If('w') {
 SetPosition(250,250);
 }
}

Start;
Repeat until ('a') {
 Move(right);
 If('x') {
 SetSpeed(1);
 }
}

```

- Funcionan como se espera, sin embargo, si la ultima tecla es 'w', el movimiento diagonal se desajusta, ya que mientras que el programa 1 hace dos movimientos (move + position) el programa 2 solo realiza 1, de modo que realiza sus acciones mas rápido, y se descoordina de este modo

## 5.4. Panel de Control

### 5.4.1. Botón Load Robot

- carga una imagen desde una ruta válida y la establece como imagen del Robot — OK
- en principio no hay restricciones de tamaño ni formato de la imagen — OK
  - valorar la posibilidad de establecerlas

### 5.4.2. Botón Load Background

- carga una imagen desde una ruta válida y la establece como imagen del mapa del Escritorio — OK
- en principio no hay restricciones de tamaño ni formato de la imagen — OK
  - valorar la posibilidad de establecerlas (para los mapas abiertos)
- hay dos tipos de imágenes que pueden ser cargadas — OK
  - **Mapas Cerrados:** son imágenes creadas ex profeso para la aplicación, las cuales constan de dos imágenes asociadas:
    - **mapa visible:** es una imagen de 512x512 px , creada con casillas o tiles de 32x32 px, con un camino que lleva de la salida a la meta, y en el que puede haber otras salidas, o trampas, u otros elementos
    - **mapa negativo:** es el mapa asociado al mapa visible, a través del nombre (mismo nombre que el visible, pero con «Negative\_» como prefijo), el cual determina mediante colores cual es el camino válido (tiles en blanco), donde esta la salida, la meta, los diferentes objetos (si los hubiere) y el resto son tiles en negro (posiciones no válidas)
  - **Mapas Abiertos:** son el resto de imágenes que pueden ser cargadas, las cuales no tienen asociado un mapa negativo, sino que comparten un mapa Negative\_default, el cual es enteramente blanco, para dejar libertad total de movimiento (hasta los bordes)

### 5.4.3. Botón Start

- al presionarlo inicia todos los programas presentes en el Escritorio — OK
- solo funciona si los programas no están parados — OK
  - esto pueden ocurrir mediante la pulsación del botón Stop, o cuando el Robot ha tocado un limite del mapa (ya sea abierto o cerrado) y ha vuelto a su posicion inicial despues de pararse (stop + reset)

- si se pulsa varias veces seguidas se ejecutan varias copias (se crean nuevos hilos) de los programas del Escritorio. Este no es el comportamiento esperado y sería necesario modificarlo.

#### 5.4.4. Botón Stop

- actúa como un interruptor con dos estados que detiene la ejecución de los programas e impide que se ejecuten mientras esté en modo stop, y reanuda y permite la ejecución en modo resume — OK
- modo Resume: en este modo (el inicial por defecto) se reanuda y permite la ejecución de los programas y muestra el ícono del siguiente estado (modo Stop) al que se pasará cuando se pulse de nuevo — OK
- modo Stop: en este modo se detiene y prohíbe la ejecución de los programas y se muestra el ícono de estado siguiente (modo Resume) — OK

#### 5.4.5. Botón Reset

- solo funciona en el modo Stop — OK
- si estamos en modo Stop, se posiciona al Robot en su posición inicial (varía según mapas) — OK

#### 5.4.6. Botón Clean

- limpia el Escritorio, eliminando todos los comandos gráficos, así como los programas lógicos creados
- borra todos los comandos gráficos, ya estén sueltos, o acoplados en un programa — OK
- elimina todos los programas lógicos — OK

#### 5.4.7. Sensor Última Tecla

- muestra la última tecla presionada — OK
  - muestra correctamente caracteres, teclas numéricas, y algunos símbolos especiales
  - no muestra algunas otras teclas (no presentes en los comandos sensores de tecla): espacio, enter, etc..

**5.4.8. Sensor Color Actual Robot**

- muestra el color de la posición actual del Robot, obtenido del pixel correspondiente a la posición actual del robot (centrado en el tile; (posición.x + 32/2, posición.y + 32/2)) — OK
- al iniciar el entorno el color actual que muestra es blanco (255,255,255) por defecto
- valorar el uso de algún tipo de imágenes (orbes, señales, globos .... ) para indicar el color de una casilla completa (no solo el pixel que comprueba el robot)

**5.4.9. Sensor Coordenadas del Ratón**

- muestra la coordenada (x,y) en que se ha hecho click dentro del Escenario, para facilitar el SetPosition — OK
- al iniciar el Entorno muestra un mensaje por defecto (coords)

**5.4.10. Sensor Color del Ratón**

- similar al indicador anterior, sin embargo este muestra el COLOR de la coordenada del Escenario donde se ha hecho click por ultima vez — OK
  - se usa para facilitar el comando Sensor de Color
- al iniciar el Entorno muestra un color por defecto (255,255,255)

**5.5. Mapas Cerrados**

- los mapas se cargan correctamente — OK
  - MAPA VISIBLE: se cargan correctamente todos los mapas cerrados creados hasta el momento - OK
  - MAPA NEGATIVO: los mapas negativos se cargan correctamente, ya que cada uno tiene el mismo nombre que el visible asociado y ubicado en el mismo directorio - OK
  - ROBOT EN SALIDA: el robot se posiciona correctamente en la salida en cada caso (tile azul) - OK
    - pivot: la imagen del Robot se ubica en el centro de la casilla — OK
  - PORTAL EN META: el portal de salida se ubica correctamente en cada caso (tile verde) - OK
    - pivot: la imagen del Portal se ubica correctamente en el centro de la casilla — OK

- ITEMS: está pendiente la posibilidad de añadir mas elementos a los mapas, como trampas, indicadores de color, enemigos, etc, los cuales se marcarán de sus propios colores en el mapa negativo, y se almacenarán en un array de mapItems en el mapa, en función del cual los pintará el Escenario
- el Robot se comporta correctamente
  - COLISIONES: cuando se mueve a una casilla negra (fuera del camino), se activa una animación de explosión y vuelve a la casilla de salida
  - SALIDA: cuando llega a la casilla del Portal debe activarse una animación de teletransporte, y quizás cargar el siguiente mapa
  - OTRAS INTERACCIONES: posibilidad de que el robot interaccione con otras entidades
  - SETPOSITION:
    - Ok para posiciones legales (dentro del camino no-negro)
    - No válido para posiciones ilegales (tiles negras)
  - SENSORCOLOR: comprueba el pixel central del tile (posicion.x + tileSize / 2, posicion.y + tileSize / 2)

## 5.6. Mapas Abiertos

- los mapas se cargan correctamente – OK
  - MAPA VISIBLE: se puede cargar cualquier imagen como mapa visible, sin embargo esto influye en el tamaño del Escenario y por tanto en el tamaño del resto de paneles de la GUI, de modo que se recomienda usar imágenes de tamaños no muy grandes ni muy pequeños
  - MAPA NEGATIVO POR DEFECTO (NEGATIVE DEFAULT): se crea en función de las dimensiones del mapa visible, de modo que se pueden usar imágenes de cualquier tamaño y el mapa negativo se adaptará a ellas (entero en blanco)
  - ERROR DE LECTURA: en caso de que no se pueda leer la imagen del archivo, se cargará un mapa en blanco de 512x512 px como mapa abierto visible (así como para el mapa negativo por defecto)
  - POSICION INICIAL: la posición inicial está por defecto en (256,256), ya que es el punto medio para el caso de imágenes de 512x512 px, que son el estándar de medición
- el Robot se puede mover libremente en cualquier dirección hasta tocar el borde del mapa (borde de 1 tile o 32 px) — OK

## 6 Despliegue

- Para la instalación de la aplicación, basta con descomprimir el archivo en cualquier directorio, siempre que el directorio de Imágenes (que contiene las imágenes básicas usadas por la aplicación) esté en la misma ruta que el ejecutable.
- En el directorio del Ejecutable hay una carpeta de imágenes básicas usadas por la aplicación (comandos, botones, etc.) que debe encontrarse en el mismo directorio que el ejecutable en .jar, ya que las rutas se han establecido de forma relativa para hacer referencia a esta carpeta de imágenes.
- Asimismo se ha incluido un ejecutable nativo de windows creado a partir del .jar con JSmooth, el cual debe estar en el mismo directorio que el .jar, ya que obtiene el classpath y la clase principal de forma relativa a este directorio.
- Para importar el proyecto en Eclipse con fines de ejecución y prueba basta con cargar la carpeta Arquitectura/PFG dentro del entorno de desarrollo. Dentro de este directorio se encuentra toda la arquitectura de la aplicación utilizada para el desarrollo, incluyendo el código fuente, las imágenes necesarias, la documentación del proyecto, la API de la versión actual (0.11) y los diversos diagramas UML.
- En las diversas carpetas dentro del directorio «Documentación PFG» de la Arquitectura, que hacen referencia a las áreas del UP, se han eliminado los documentos en LyX creados durante el desarrollo (Modelo de Casos de Uso, Especificación Complementaria, Modelo de Diseño, etc.) para documentar el trabajo. Esto es debido a que la mayoría de esos documentos individuales se han agregado para conformar la presente memoria.

# **7 Gestión del Proyecto**

## **7.1. Plan de Proyecto**

### **7.1.1. Introducción**

El objetivo es desarrollar una aplicación educativa que permita a niños de primaria familiarizarse con la programación y desarrollar sus primeros programas, aunque también se prevé la posible ampliación de la aplicación para su uso por parte de cualquier persona no familiarizada con la programación, dentro de cualquier rango de edades.

En el presente documento se detallan los aspectos generales del proyecto, así como el calendario general previsto inicialmente, el cual puede estar sujeto a modificaciones según avance el desarrollo del proyecto.

Asimismo, se especifica como se va a organizar el proyecto, tanto en el ámbito de la planificación y desarrollo, como a nivel documental. También se definirá la metodología y las herramientas de desarrollo que se van a utilizar.

### **7.1.2. Organización del Proyecto**

El proyecto se va a organizar siguiendo el esquema de la metodología de desarrollo iterativo del Proceso Unificado, de modo que la estructura de directorios prevista se dividirá en áreas de desarrollo de la siguiente forma:

- 01 Modelado de Negocio
- 02 Requisitos
- 03 Diseño
- 04 Implementacion
- 05 Pruebas
- 06 Despliegue
- 07 Gestión del Proyecto
- 08 Otros
- 09 Control de Versiones

En la tabla que se muestra a continuación, se detalla que documentos y artefactos del UP se engloban dentro de cada una de las áreas anteriores, así como en que fases del desarrollo se trabaja sobre cada una:

Cuadro 7.1: Artefactos por áreas en el UP

| Área                 | Artefacto            | Fases del Desarrollo |             |              |            |
|----------------------|----------------------|----------------------|-------------|--------------|------------|
|                      |                      | Inicio               | Elaboración | Construcción | Transición |
| Modelado de negocio  | Modelo de dominio    | c                    | r           |              |            |
| Requisitos           | Visión               | c                    | r           |              |            |
|                      | Modelo Casos de Uso  | c                    | r           |              |            |
|                      | Especificación Compl | c                    | r           |              |            |
|                      | Glosario             | c                    | r           |              |            |
| Diseño               | Modelo de Diseño     |                      | c           | r            |            |
|                      | Arquitectura SW      |                      | c           | r            |            |
|                      | Modelo de Datos      |                      | c           | r            |            |
| Implementación       | Modelo de Implem.    |                      | c           | r            | r          |
| Pruebas              | Modelo de Pruebas    |                      | c           | r            | r          |
| Despliegue           | Plan de Despliegue   |                      |             |              | c          |
| Gestión de Proyecto  | Plan de Proyecto     | c                    | r           | r            | r          |
|                      | Plan de Iteracion    | c                    | r           | r            | r          |
| Otros                | *                    | ?                    | ?           | ?            | ?          |
| Control de Versiones | **                   | c                    | r           | r            | r          |

- \* : en este área se englobarán en principio todos aquellos documentos que no sean aplicables a ninguna otra disciplina del UP
- \*\* : para el control de versiones, se tomarán «fotografías» del conjunto del proyecto, documentación y código por igual, al final de cada iteración del desarrollo, de modo que no existirá en principio un documento específico para ello, sin perjuicio de que se pueda crear uno posteriormente donde se recojan los cambios en cada iteración (changelog)

### 7.1.3. Prácticas y medidas del Proyecto

El proyecto se llevará a cabo siguiendo la metodología de desarrollo / ciclo de vida iterativos, concretamente, y dado que se va a seguir un enfoque orientado a objetos, se seguirá la metodología de desarrollo del Proceso Unificado (UP). Se opta por este tipo de desarrollo dado que el UP engloba procedimientos de desarrollo comúnmente aceptados como «buenas prácticas», tales como ciclo de vida iterativo, desarrollo dirigido por el riesgo, de modo que se pueden abordar rápidamente los aspectos mas cruciales de la aplicación, permitiendo reducir la incertidumbre y probar la aplicación de forma temprana.

Dentro de este proceso de desarrollo se estructurará el desarrollo en 4 fases, Inicio, Elaboración, Construcción y Transición, cada una de las cuales se estructurará en una o varias iteraciones de duración fija, dentro de las que se irá refinando progresivamente el desarrollo del sistema.

En cada una de las iteraciones, y dependiendo de en qué fase nos encontremos, se tratarán en mayor o menor medida todos los aspectos de desarrollo, análisis, diseño, implementación, pruebas e integración. En la siguiente figura se muestran algunos de los diferentes artefactos del proceso de desarrollo y su interrelación.

Figura 7.1: Muestra de artefactos del UP

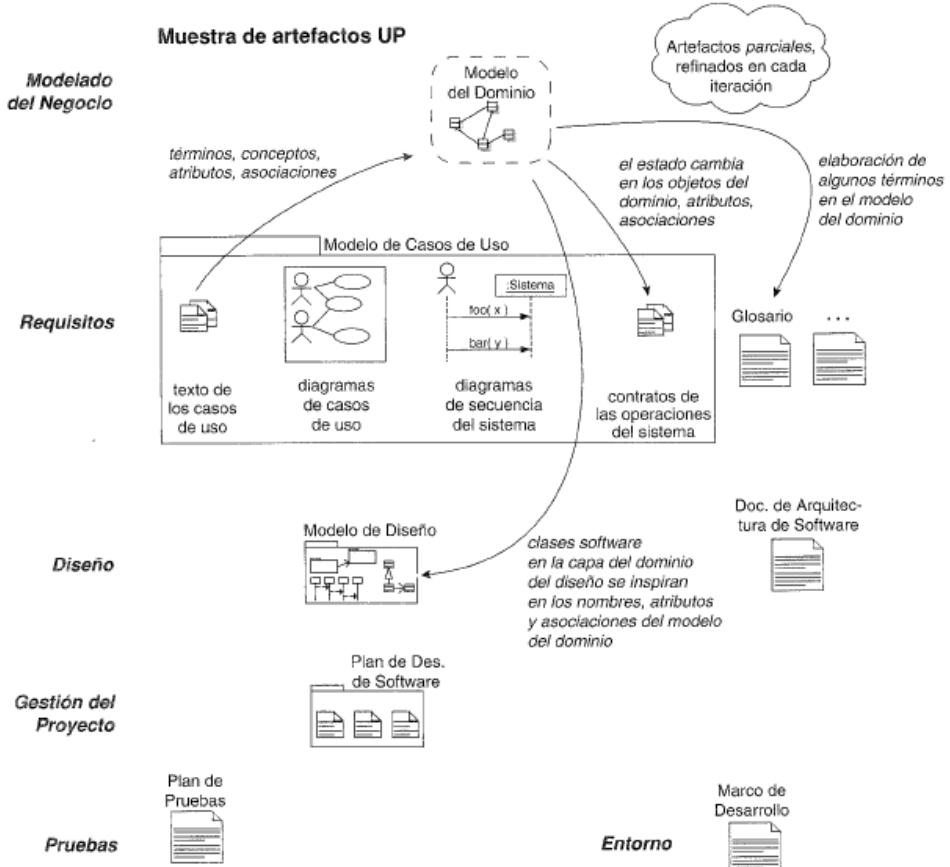


Figura 10.12. Muestra de la influencia entre los artefactos UP.

En el apartado siguiente se muestra un calendario inicial del proyecto, en el cual se detallan las fases comentadas y sus iteraciones, especificando fechas e hitos de desarrollo. El mencionado calendario se establece en base al análisis inicial llevado a cabo en la fase de inicio, por lo cual podría estar sujeto a cambios, que se tratarán de minimizar,

conforme avance el desarrollo del software.

En cuanto a la selección de las herramientas de desarrollo, se prevé el uso de las siguientes:

- se utilizará el entorno Eclipse (v. Juno) para el desarrollo, ya que el mismo tiene soporte para la utilización del UP, así como guías de desarrollo bajo dicha metodología (OpenUP )
  - asimismo, se utilizará el entorno NetBeans para el desarrollo de la GUI, ya que permite una gestión más visual de las mismas, y la aplicación tiene mucho componente de interfaz en su desarrollo
- para la generación de la documentación se usará generalmente L<sub>X</sub>X, debido a su flexibilidad y a su facilidad de conversión de documentos a pdf
- el entorno de implementación y pruebas será en principio el SO Windows 7
- se utilizará el software de modelado UML Astah para la realización de diagramas UML.
  - se utiliza asimismo el plugin de Eclipse ObjectAidUML para la generación de diagramas UML desde código fuente
- se utiliza IllustStudio y Photoshop como software de edición de imágenes para crear y editar las imágenes de los comandos y el resto de imágenes del entorno

#### 7.1.4. Calendario General del Proyecto

Cuadro 7.2: Calendario General del Proyecto

| Fase         | Iteración | Objetivos Principales                                                                                                                                                                    | Fechas inicio / fin | Duración |
|--------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|----------|
| Inicio       | I1        | Planificación inicial<br>Calendario general<br>Requisitos iniciales (75 % UC formato breve)<br><br>Establecimiento herramientas                                                          | 17-feb / 3-mar      | 15 d     |
| Elaboracion  | E1        | definir UC detalle (10 %)<br>definir Modelo dominio<br>comenzar diseño de objetos, diagramas de clase<br>comenzar la implementacion UC (10 %)<br><br>probar UC (10 %)                    | 3-mar / 17-mar      | 15 d     |
| Elaboracion  | E2        | definir UC detalle (25 %)<br>refinar Modelo de dominio<br>refinar diseño de objetos, diagramas de clase<br>definir Arquitectura SW<br>implementacion UC (25 %)<br><br>probar UC (25 %)   | 17-mar / 31-mar     | 15 d     |
| Elaboracion  | E3        | definir UC detalle (40 %)<br>completar Modelo de dominio<br>refinar diseño de objetos, diagramas de clase<br>refinar Arquitectura SW<br>implementacion UC (40 %)<br><br>probar UC (40 %) | 31-mar / 14 abr     | 15 d     |
| Construcion  | C1        | revisar Modelo UC (90 %)<br>refinar DCD<br>refinar Arquitectura SW<br>implementación UC (50 %)<br><br>pruebas UC (50 %)                                                                  | 14-abr / 28 abr     | 15 d     |
| Construcción | C2        | requisitos terminados en su mayoria<br>refinar DCD<br>refinar Arquitectura SW<br>implementación UC (75 %)<br>pruebas UC (75 %)<br><br>iniciar Memoria                                    | 28-abr / 12-may     | 15 d     |
| Construcción | C3        | refinar DCD<br>refinar Arquitectura SW<br>implementación UC (75 %)<br>pruebas UC (75 %)<br><br>refinar Memoria                                                                           | 19-may / 2-jun      | 15 d     |
| Construcción | C4        | completar Arquitectura SW<br>implementación UC (85 %)<br>pruebas UC (85 %)<br>iniciar pruebas de integración<br>iniciar pruebas de usabilidad<br><br>refinar Memoria                     | 2-jun / 16 jun      | 15 d     |
| Construcción | C5        | completar Arquitectura SW<br>implementación UC (90 %)<br>pruebas UC (90 %)<br>refinar pruebas de integración<br>refinar pruebas de usabilidad<br><br>refinar Memoria                     | 16-jun / 30 jun     | 15 d     |
| Transición   | T1        | completar pruebas integración <sup>94</sup><br>finalizar pruebas de usabilidad<br>diseño de mapas<br>iniciar documentación final (MU, MP)<br><br>refinar Memoria                         | 30 jun / 14 jul     | 15 d     |
| Transición   | T2        | implantación<br>pruebas finales de conjunto<br>completar documentacion final<br><br>completar Memoria                                                                                    | 14 jul / 28 jul     | 15 d     |

## 7.2. Planes de Iteración

A continuación se detallan los Planes de Iteración, elaborados para cada iteración durante el desarrollo de la aplicación, en orden cronológico descendente. Cada plan de iteración se establece al inicio de cada iteración y se valora cuando se finaliza la misma, de modo que los aspectos abiertos y problemas encontrados sirvan de base para la elaboración de la siguiente iteración. En cada uno de ellos se detallan los siguientes aspectos:

- la iteración a la que pertenece
- los objetivos de alto nivel que se consideraron para dicha iteración
- los items de trabajo específicos sobre los que se pretendía trabajar, detallando una estimación del esfuerzo previsto, el área al que pertenecen y una sección de comentarios
- los problemas encontrados y aspectos abiertos al finalizar la iteración
- una valoración del trabajo realizado y pendiente para la siguiente iteración

### 7.2.1. Plan de Iteración T2

#### 7.2.1.1. Iteración Actual

- Transición 2 (T2)

#### 7.2.1.2. Objetivos de Alto Nivel

- implantación
- finalizar pruebas de funcionalidad
- finalizar pruebas de usabilidad
- diseño de mapas cerrados
- completar documentación final (MU, MP)
- completar Memoria

### 7.2.1.3. Items de Trabajo

Cuadro 7.3: Items de trabajo iteración T2

| Descripción                                                                                                                                                          | Esfuerzo estimado | Área       | Notas |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------|-------|
| Diseño fondos / robots (sprites)                                                                                                                                     | 20h +             | Dis, Impl  |       |
| actualizar Arquitectura<br>- diagramas UML explicativos                                                                                                              | 2h                | Dis        |       |
| presentación<br>- validar look & feel<br>- colores, menus, nombres, etc.                                                                                             | 5h                | Despl      |       |
| Documentación Final<br>- Manual de Usuario (pdf / avi)<br>- Manual Instalación<br>- Modelo de Pruebas (func / usab)<br>- Sección Implementación<br>- Revisar Memoria | 20h +             | Pru, Despl |       |
| implantación<br>- exportar a .jar<br>- pruebas con ejecutable                                                                                                        | 2h                | Despl      |       |
| valoración iteración (T2)                                                                                                                                            | 1h                | G.Pr       |       |
| changelog y aspectos abiertos                                                                                                                                        | 1h                | Otros      |       |

### 7.2.1.4. Problemas y Aspectos Abiertos

| Problema             | Estado        | Nota |
|----------------------|---------------|------|
| Manual Usuario (avi) | pendiente     | 1    |
| Mapas / Sprites      | en desarrollo | 2    |
|                      |               |      |

1. Quizás se podría realizar algún vídeo explicativo con ejemplos para obtener una ayuda mas visual al Manual de Usuario, que fuera mas comprensible y amena para los alumnos
2. Pendiente de ampliar el repertorio de Mapas Cerrados, añadiéndoles mas diversidad, mas elementos, etc

### 7.2.1.5. Valoraciones

- se ha ajustado la lógica del snap:
  - ANTES: iteraba sobre todos los comandos gráficos del Escritorio comprobando secuencialmente con cual se podía acoplar, en función de si se intersectaban y si era un snap legal, de acuerdo con las reglas de la gramática, y se acoplaba al primero que cumplía las condiciones

- AHORA: en función del punto de drop, y el rectángulo de la imagen del comando, explora las intersecciones de todos los comandos gráficos del Escritorio con el comando que se desea acoplar, y la intersección (rectángulo) con el mayor área (width \* height) es el ganador, formando el comando objetivo del snap
- se ha ajustado la lógica del STOP
  - switchStopButton: ahora este método cambia no solo la imagen del botón, sino también el estado del Panel de Control, que influye en las acciones de parar programas (ya sea por botón Stop o por movimiento ilegal)
  - stopped: se usa un atributo booleano para simular el estado del Panel de Control
  - moveRobot: ahora no se producen desajustes cuando existen varios programas que mueven al Robot en varias direcciones simultáneamente, ya que el evento de Stop que se produce cuando algún programa hace un movimiento ilegal, ahora detiene todos los programas e impide la recepción de sucesivos eventos del mismo tipo (hasta que se pulse el botón Stop)
  - de este modo la lógica del stop es mas consistente y menos propensa a fallos
- MU y MP: se ha redactado el Manual de Usuario y se ha completado el Manual de Pruebas
- Memoria: se han completado y actualizado las secciones pendientes de la Memoria
- UML: se han actualizado los diagramas UML de la aplicación y el diagrama de la Arquitectura lógica
- Se ha probado la aplicación desde el ejecutable, y parece que funciona como se espera, siempre que la carpeta de imágenes esté en el mismo directorio del ejecutable

### 7.2.2. Plan de Iteración T1

#### 7.2.2.1. Iteración Actual

- Transición 1 (T1)

#### 7.2.2.2. Objetivos de Alto Nivel

- completar pruebas integración
- finalizar pruebas de usabilidad
- diseño de mapas
- iniciar documentación final (MU, MP)
- refinar Memoria

### 7.2.2.3. Items de Trabajo

Cuadro 7.4: Items de trabajo iteración T1

| Descripción                                                                                                        | Esfuerzo estimado | Área              | Notas          |
|--------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|----------------|
| - pruebas gestión de comandos lógicos<br>- eliminar comandos                                                       | 10h               | Dis, Impl,<br>Pru |                |
| JComboBox<br>- funcionalidad de <i>pipeta</i>                                                                      | 4h                | Dis, Impl,<br>Pru | Baja Prioridad |
| Acciones Robot<br>- pruebas extensivas<br>- pruebas con mapas abiertos                                             | 10h +             | Pru               |                |
| Diseño fondos / robots (sprites)                                                                                   | 10h +             | Dis, Impl         |                |
| actualizar Arquitectura<br>- diagramas UML explicativos                                                            | 2h                | Dis               |                |
| Documentación Final<br>- Manual de Usuario<br>- Modelo de Pruebas<br>- Sección Implementación<br>- Revisar Memoria | 10h +             | Despl             |                |
| valoración iteración (T1)                                                                                          | 1h                | G.Pr              |                |
| changelog y aspectos abiertos                                                                                      | 1h                | Otros             |                |
| definir Plan de Iteración siguiente<br>(T2)                                                                        | 1h                | G.Pr              |                |

### 7.2.2.4. Problemas y Aspectos Abiertos

| Problema                | Estado        | Nota |
|-------------------------|---------------|------|
| mapas cerrados / robots | en desarrollo | 1    |
| mapas abiertos          | en desarrollo | 2    |
| .jar                    | pendiente     | 3    |
| arquitectura / UML      | en revisión   | 4    |
| documentación           | en desarrollo | 5    |

1. se han diseñado unos pocos niveles de mapas cerrados de muestra, pero tiene mucho potencial de extensibilidad:
  - a) enemigos, objetos, historia, congrats en meta, cambio de nivel ...
  - b) orbes / señales de color (sensor Color)
  - c) mapas temáticos para usar los comandos específicos
2. pendiente el uso correcto del defaultNegativeMap en mapas abiertos (ojo con los portales, que no deben aparecer)

3. pendiente verificar la carga de imágenes cuando se exporte la aplicación al ejecutable (subdirectorio?)
4. pendiente revisar y actualizar arquitectura y diagramas UML
5. pendiente terminar Manual/Plan de Pruebas, Manual de Usuario, Manual de Instalación (en caso de ser necesario) y Memoria con ejemplos (quizás en el MU)

#### 7.2.2.5. Valoraciones

- se ha implementado una solución sencilla para la eliminación de los comandos del Escritorio, incluyendo un botón de limpiar en el P.Control, el cual elimina todos los comandos gráficos y los programas lógicos creados del Escritorio
- se han incluido dos indicadores mas en el P.Control, Coordenadas del Mouse, y Color del Mouse, los cuales permiten registrar las coordenadas (x,y) y el color (respectivamente), de la ultima posición del Escenario donde se haya hecho click
  - el MouseCoords facilita el uso del comando SetPosition
  - el MouseColor simula la funcionalidad de pipeta, indicando el color de una determinada posicion del mapa, para poder usar el Comando Sensor de Color
- clase Map: toda la lógica de creación de mapas ha sido encapsulada en el propio objeto Map contenido por el Escenario
  - libera de las responsabilidades específicas al Escenario
- se ha añadido la clase MapManager (factoría) con facilidades de creación de mapas
- se han añadido los atributos rightSlot & downSlot en los Comandos para evitar acoplar varios comandos al mismo (snap mas robusto ahora)
- se ha cambiado Sensor para que sea una clase abstracta (en vez de una interfaz) que hereda de Sent. De este modo se puede mover el atributo logicalCom (de tipo Sent) a la superclase Comando, y dejar que cada subclase le cree según su semántica, y mas importante, se puede recuperar el comando lógico de un comando gráfico objetivo (el objetivo del snap), para que la creación de programas lógicos sea mas robusta y transparente, y no tener que iterar sobre todo el programa, ya que con el snap sabemos a que comando gráfico le estamos acoplando otro (el comando fuente)
  - de momento se usa para el acoplamiento de los Sensores en los comandos de control (Repeat, Wait, If)
  - valorar usarlo tb para el resto del snap
- debido a lo anterior, ya no es necesario colocar los sensores de los comandos de Control en orden según se va creando el programa; pueden colocarse en cualquier orden.

- para darle mas homogeneidad y transparencia quizás seria deseable cambiar no solo el snap de Sensores lógicos atendiendo al targetCom (gráfico) sino también hacerlo para el snap del resto de Sentencias
  - la responsabilidad de esto recae en los siguientes métodos de la clase Programa: getCurrentBloque() / closeCurrentBlock()
  - explicar esto en el MU
- Pruebas: se ha iniciado la redacción y ejecución del Plan de Pruebas cubriendo las funcionalidades mas relevantes de la aplicación
    - pendiente completar las secciones de Mapas abiertos / cerrados y depurar algunos errores

### 7.2.3. Plan de Iteración C5

#### 7.2.3.1. Iteración Actual

- Construcción 5 (C5)

#### 7.2.3.2. Objetivos de Alto Nivel

- completar Arquitectura SW
- implementación UC (90 %)
- pruebas UC (90 %)
- refinar pruebas de integración
- refinar pruebas de usabilidad
- refinar Memoria

### 7.2.3.3. Items de Trabajo

Cuadro 7.5: Items de trabajo iteración C5

| Descripción                                                                                                    | Esfuerzo estimado | Área           | Notas          |
|----------------------------------------------------------------------------------------------------------------|-------------------|----------------|----------------|
| - pruebas gestión de comandos lógicos<br>- eliminar comandos                                                   | 10h               | Dis, Impl, Pru | Alta prioridad |
| JComboBox<br>- colores de ComandoColor<br>- obtener colores de escenario                                       | 10h               | Impl, Pru      |                |
| Sentencia Wait<br>- gestión de hilos (Executor)<br>- control de concurrencia                                   | 8h                | Impl, Pru      |                |
| Acciones Robot<br>- probar sentencias de control<br>- documentar pruebas                                       | -                 | Pru            |                |
| Diseño fondos / robots (sprites)                                                                               | -                 | Dis, Impl      | Baja prioridad |
| UC5 - 6 - 7 (Stop y Reset)<br>- completar DI's<br>- implementar acciones P.Control<br>- realizar y doc pruebas | 4h                | Dis, Impl, Pru | Baja prioridad |
| actualizar Arquitectura<br>diagramas UML explicativos                                                          | 2h                | Dis            |                |
| modelo de UC<br>- revisar acciones lógicas (CO5)                                                               | 2h                | Req            |                |
| Refinar Memoria<br>- apartado impl<br>- apartado pruebas                                                       | 2h                | Despl          |                |
| valoración iteración (C5)                                                                                      | 1h                | G.Pr           |                |
| changelog y aspectos abiertos                                                                                  | 1h                | Otros          |                |
| definir Plan de Iteración siguiente<br>(C6 o T1)                                                               | 1h                | G.Pr           |                |

### 7.2.3.4. Problemas y Aspectos Abiertos

| Problema                              | Estado        | Nota |
|---------------------------------------|---------------|------|
| Program Container                     | en revisión   | 1    |
| Robot Mover                           | pendiente     | 2    |
| eliminar comando (UC3)                | pendiente     | 3    |
| diseño mapas                          | en desarrollo | 4    |
| arquitectura sw                       | en revisión   | 5    |
| Manual de usuario (MU) / pruebas (MP) | pendiente     | 6    |
| ComandoColor                          | en revisión   | 7    |

1. de momento es el Escritorio el que obtiene la lista de programas del pc y se encarga de ejecutarlos (mediante el ProgramExecutor) dentro de su propio metodo (onPropertyEvent())
  - a) quizás a efectos de mejorar la cohesión debería delegarse completamente en el pc
2. seria mejor (desde el p.vista de la cohesión) que el Escenario tuviera un att al estilo del programContainer para gestionar la mecánica NO-GRAFICA ?
3. pendiente de diseñar e implementar una solución para la eliminación de comandos del Escritorio
4. pendiente refinar el diseño de mapas, aplicando colisiones, niveles, etc
5. se debe revisar el documento de arquitectura para que sea consistente con los últimos cambios, y ver si se puede mejorar la cohesión / acoplamiento / extensibilidad
6. pendiente de realizar la documentación final del proyecto (Manual Usuario, Manual de Pruebas, Memoria definitiva con ejemplos / pantallazos, etc.)
7. valorar la necesidad o utilidad de añadir la funcionalidad de «pipeta» al comandoColor, para obtener los colores del Escenario

#### 7.2.3.5. Valoraciones

- Sent\_Wait: se ha probado a implementar la semántica con un bucle de espera ocupada y la solución parece que funciona correctamente, incluso con varios programas en el escritorio (ya que cada uno es controlado por un hilo diferente)
  - esta solución ocupa tiempo de procesador, y puede ser ineficiente en rendimiento si existen muchos programas en el Escritorio corriendo a la vez
  - sin embargo, aunque virtualmente se pueden crear un n° ilimitado de programas, en la práctica este número está limitado por el espacio gráfico del Escenario
  - no obstante, si se quisiera extender la solución para permitir un n° elevado de programas debería implementarse mediante un wait() / notify()
- Sent\_Wait: se ha realizado otra implementación alternativa de la semántica del wait, usando los metodos wait() y notify() de java.Thread, reduciendo de este modo el consumo de ciclos de CPU que se daba con la implementación de la espera ocupada anterior.
  - se ha probado la solución del wait - notify y parece que funciona correctamente (pendiente de pruebas mas extensivas)

- tal como está implementada se llama al wait del hilo de cada programa contenido de una sentencia wait, y para despertar al hilo se llama a un método wakeUP en cada sentencia de cada programa, los cuales son vacíos excepto en el caso de las sentencias If y Repeat, que llaman recursivamente a los wakeUP de sus propios bloques de sentencias, y en el caso de la propia sentencia Wait, en el cual se comprueba si coincide su sensor con el del P.Control, y en caso afirmativo se llama al notify() del hilo de ese programa
- Se ha cambiado la forma de acceso de los comandos lógicos (actuadores Move, SetPosition y SetSpeed) para que se haga desde el único punto de acceso al singleton del Entorno, y desde ahí obtengan el Escenario, sobre el cual llamen a sus métodos de movimiento del robot
  - de este modo se cambia al patrón Experto, ya que el Escenario es el que tiene toda la información sobre el Robot y sobre el Fondo, y sobre todo, es el que sabe cuáles son las posiciones legales sobre el Fondo (negativeMap)
  - además de esta forma se impide que los comandos manipulen directamente al Robot, respetando así el principio de Encapsulamiento.
- Se ha implementado una solución de Stop / Resume (UC6 y 7) basada en flags booleanos y guardas en las Sentencias
  - se ha cambiado Sent de interfaz a clase abstracta para incluir atributos y métodos basados en esta solución (*stopped*)
- Se ha implementado una solución de Reset, que devuelve el Robot a su posición inicial, solo si los programas están parados
- El Panel de Control tiene ya toda la funcionalidad completa
- se ha implementado la elección de colores de los comandos Color mediante la lista desplegable del comboBox y mediante un ColorChooser pinchando en el recuadro de color del comando. Los colores son objetos de clase java.awt.Color, y ya no son imágenes.
  - pendiente valorar la inclusión de una funcionalidad de «*pipeta*» para obtener los colores del fondo de Escenario
- En principio las acciones del Robot están implementadas completamente, aunque falta someterlo a un plan de pruebas más extensivo, y probarlas en relación a los nuevos mapas que se diseñen (mapas con niveles, colisiones, enemigos, objetivos, etc.)

## 7.2.4. Plan de Iteración C4

### 7.2.4.1. Iteración Actual

- Construcción 4 (C4)

### 7.2.4.2. Objetivos de Alto Nivel

- completar Arquitectura SW
- implementación UC (85 %)
- pruebas UC (85 %)
- iniciar pruebas de integración
- iniciar pruebas de usabilidad
- refinar Memoria

### 7.2.4.3. Items de Trabajo

Cuadro 7.6: Items de trabajo iteración C4

| Descripción                                                                                     | Esfuerzo estimado | Área                | Notas          |
|-------------------------------------------------------------------------------------------------|-------------------|---------------------|----------------|
| Panel de Control<br>- añadir ult. tecla / color<br>- relaciones con los Comandos Sensores       | 8h                | Req, Dis, Impl, Pru | Prerrequisito  |
| - gestión de comandos lógicos<br>- eliminar comandos                                            | 10h               | Dis, Impl, Pru      | Alta prioridad |
| JComboBox<br>- cambiar valores en tº ejecución<br>- colores de ComandoColor                     | 8h                | Impl, Pru           |                |
| Acciones Robot<br>- probar sentencias de control<br>- documentar pruebas                        | 4h                | Impl, Pru           |                |
| Diseño fondos / robots (sprites)                                                                | 20h+              | Dis, Impl           | Baja prioridad |
| UC5 - 6 - 7<br>- completar DI's<br>- implementar acciones P.Control<br>- realizar y doc pruebas | 4h                | Dis, Impl, Pru      | Baja prioridad |
| modelo de UC<br>- revisar acciones lógicas (CO5)                                                | 2h                | Req                 |                |
| Refinar Memoria<br>- modelo de negocio ?<br>- apartado impl<br>- apartado pruebas               | 2h                | Despl               |                |
| valoración iteración (C4)                                                                       | 1h                | G.Pr                |                |
| changelog y aspectos abiertos                                                                   | 1h                | Otros               |                |
| definir Plan de Iteración siguiente (C5 o T1)                                                   | 1h                | G.Pr                |                |

#### 7.2.4.4. Problemas y Aspectos Abiertos

| Problema       | Estado      | Nota |
|----------------|-------------|------|
| Color / Key    | en revisión | 1    |
| UML            | pendiente   | 2    |
| acoplamiento   | en revisión | 3    |
| sents. control | en revisión | 4    |
| concurrency    | pendiente   | 5    |
| sent Wait      | en revisión | 6    |
|                |             |      |

1. las implementaciones de estas clases se hacen extendiendo las de Java (security.Key, y java.awt.Color)
  - a) buscar una forma mas limpia de hacerlas ...
  - b) podrían usarse las de Java directamente ??
  - c) comandoColor:
    - 1) Color: usar colores en el comando Sensor en vez de imágenes
    - 2) obtener los colores del escenario según la posición del cursor (pipeta similar a software dibujo o scratch)
2. es necesario hacer mas diagramas UML que expliquen la lógica interna de la aplicación (no solo en relación a la Memoria, sino tb a efectos de seguimiento y claridad) y actualizar el diagrama de Arquitectura SW para reflejar los nuevos cambios
3. ver si hay demasiadas dependencias en relación al manejo de eventos del escenario y el panel de control y tratar de reducir el acoplamiento en lo posible
  - a) ver si se pueden unificar las interfaces de gestión de eventos
  - b) revisar diagramas de clases (DCD) y de interacción (DI)
4. la estimación del esfuerzo de las pruebas de las acciones del robot y de las sentencias de control es demasiado optimista. Será necesario invertir mas horas en la prueba, ya que se trata realmente de las pruebas del sistema final
5. Concurrencia: el sistema debe poder ejecutar concurrentemente todos los programas presentes en el escritorio,
  - a) se necesitará usar un hilo para cada programa, ya que si hay una sentencia wait en uno (por ejemplo), se dormirán todos en la implementación con un solo hilo
  - b) para desacoplar la gestión de concurrencia del resto de la aplicación, sería conveniente usar un Executor del paquete java.concurrent, que centralice la creación de hilos / programa
6. Sentencia Wait: ver como se hace la implementación de la acción semántica

- a) en principio se opta por hacer que el hilo del programa lógico al que pertenece espere (hacer un Thread.wait()), hasta que cambie un sensor del P.Control, el cual debe despertar a dicho hilo (mediante un notify()) - ver como realizamos esto
  - 1) el programa debe recibir el aviso, comprobar que los sensores coinciden y despertarse en su caso (el *notify()* como tal le despierta directamente: probar esto)
- b) valorar otras soluciones alternativas

#### 7.2.4.5. Valoraciones

- Se ha modificado el 2.1.3.5 para englobar el registro de los eventos sensores (input de teclado y color de la posición del Robot)
- Se han incluido los elementos sensores del Panel de Control, *Ultima Tecla* y *Color Actual*, y se ha probado extensivamente el funcionamiento de los mismos, el cual responde correctamente.
- Se han incluido descripciones auxiliares de los botones del Panel de Control (toolTipText) que aparecen al dejar el cursor posicionado sobre ellos.
- Se ha cambiado la construcción de los comandos en la factoría, para crear con cada comando gráfico su comando lógico asociado a la vez (dentro del constructor), de este modo el ProgramContainer solo tiene que recuperar el comando lógico y añadirlo al programa lógico que se está creando
  - de este modo queda una implementación más cohesiva y menos acoplada (la creación de comandos lógicos queda encapsulada dentro del constructor del comando gráfico)
- se ha implementado un delay en los bucles para que las acciones del robot sean observables
  - Sleep(): en principio se ha implementado con un sleep y asignando un Thread para cada programa, de modo que cada programa no afecte a otros
  - se han valorado otras soluciones, si bien no funcionan tan bien como el sleep():
    - Timer de Swing (para ello debería implementar ActionListener la Sent\_repeat ...)
    - Wait()
- se han creado relaciones entre cada comando Sensor lógico (creados dinámicamente por el ProgramContainer del escritorio) con los sensores del P.Control, dentro del método *isTrue()* de cada uno de ellos (que comprueba si coincide el sensor del P.Control con el sensor lógico de la sentencia), de la siguiente forma:

- cada comando sensor lógico (sentencia) comprueba si su atributo sensor es igual al del P.Control accediendo al singleton del Entorno y desde ahí al sensor del P.Control
  - aunque existe acoplamiento ya que cada sentencia sensor debe acceder a un elemento superior en la jerarquía, este se minimiza mediante el uso del Singleton
  - no se puede usar el patron Observer, precisamente porque se crean dinámicamente y no hay forma elegante de añadir los suscriptores ...
- se ha modificado el funcionamiento de los botones del Panel de Control para que inmediatamente después de que el usuario los pulse (y se envíe el evento correspondiente a los oyentes), se devuelva el focus al Escenario, de modo que quede preparado para recibir eventos de control (basicamente los inputs de teclado, ya que los cambios de color debidos al movimiento del robot ya se realizaban correctamente antes de la modificación)
  - Se han modificado los comboBox de los Comandos (gráficos) para que se puedan cambiar en tiempo de ejecución, y afecte realmente a la semántica de los programas (y a las acciones del robot).
  - Se ha arreglado la creación de programas lógicos (comprobaciones de placed y snapped) para que creen un solo programa por cada bloque de comandos pegados que comience con un Start (comando Iniciar)
    - PENDIENTE realizar mas pruebas sobre esto, para comprobar la robustez de la solución
    - PENDIENTE la gestión de la eliminación de los comandos de un programa
  - PENDIENTE cambiar los items del comboBox del comandoColor para que sean de tipo Color (combo box y sensor logico)
    - hay que ver como podemos obtener los colores del fondo del escenario (mouseOver event - mouseEntered y mouseExited)
  - PENDIENTE la gestión de hilos (Executor) para resolver el problema de la sentencia Wait y la concurrencia en general
  - PENDIENTE diseño gráfico sprites del robot y fondos diversos
  - PENDIENTE UC 5, 6 y 7 (botón Stop y Reset, dado que Start ya funciona como se espera)

### 7.2.5. Plan de Iteración C3

#### 7.2.5.1. Iteración Actual

- Construcción 3 (C3)

### 7.2.5.2. Objetivos de Alto Nivel

- refinar DCD
- refinar Arquitectura SW
- implementación UC (75 %)
- pruebas UC (75 %)

### 7.2.5.3. Items de Trabajo

Cuadro 7.7: Items de trabajo iteración C3

| Descripción                                                                                     | Esfuerzo estimado | Área                   | Notas                              |
|-------------------------------------------------------------------------------------------------|-------------------|------------------------|------------------------------------|
| UC5 - 6 - 7<br>- completar DI's<br>- implementar acciones P.Control<br>- realizar y doc pruebas | 4h                | Dis, Impl,<br>Pru      | - metodos dummy si<br>no da tiempo |
| Panel de Control<br>- añadir ult. tecla / color<br>- relaciones con la GUI / robot              | 5h                | Req, Dis,<br>Impl, Pru |                                    |
| - gestión de comandos lógicos<br>- crear programas<br>- añadir comandos<br>- eliminar comandos  | 15h               | Dis, Impl,<br>Pru      |                                    |
| modelo de UC<br>- revisar acciones lógicas (CO5)                                                | 2h                | Req                    |                                    |
| JComboBox<br>- refinar y documentar pruebas                                                     | 5h                | Impl, Pru              |                                    |
| Acciones Robot<br>- probar solución singleton<br>- documentar pruebas                           | 10h               | Impl, Pru              | Alta prioridad                     |
| Diseño fondos / robots (sprites)                                                                | 10h               | Dis, Impl              | Prerrequisito                      |
| Memoria<br>- apartado impl<br>- apartado pruebas                                                | 2h                | Despl                  |                                    |
| Refinar presentación<br>- layout del entorno<br>- look & feel                                   | 2h                | Req, Dis,<br>Impl      | Baja prioridad                     |
| valoración iteración (C3)                                                                       | 1h                | G.Pr                   |                                    |
| changelog y aspectos abiertos                                                                   | 2h                | Otros                  |                                    |
| definir Plan de Iteración siguiente<br>(C4)                                                     | 1h                | G.Pr                   |                                    |

#### 7.2.5.4. Problemas y Aspectos Abiertos

| Problema                       | Estado        | Nota |
|--------------------------------|---------------|------|
| eliminar Comando               | pendiente     | 1    |
| sensores P.Control             | pendiente     | 2    |
| botones P.Control (UC5,6,7)    | pendiente     | 3    |
| diseño fondos / robots         | pendiente     | 4    |
| logicalCom como atributo       | en desarrollo | 5    |
| lock comGraf (snapped, placed) | en desarrollo | 6    |
| pruebas                        | en desarrollo | 7    |

1. pendiente de definir e implementar las reglas de eliminación de comandos (gráficos y lógicos) - (solo último comando, todo el programa, cualquier comando intermedio ...)
2. pendiente de implementar los sensores del Panel de Control para poder probar las acciones del robot incluyendo sentencias de control y sus correspondientes sensores
3. queda pendiente la implementación de las acciones de los botones del Panel de Control (las de Start estan prácticamente terminadas)
4. desarrollo de fondos y robots originales en IllustStudio / Photoshop
5. se está desarrollando una solución para cambiar los valores de los comboBox de los Comandos Gráficos en tiempo de ejecución (ver si es aplicable a todos)
6. pendiente de hacer operativos los atributos tipo locked de los comandos (snapped, placed, etc) para evitar moverlos una vez acoplados a un programa
7. pendiente definir el documento de pruebas formalmente

#### 7.2.5.5. Valoraciones

- Se ha refinado la implementación de los JComboBox que representan la parte editable (o desplegable) de los comandos gráficos
  - ahora se crean solo para los comandos que los necesiten (todos excepto los de tipo control)
  - ahora admiten los tipos correctos de datos para cada tipo de comando
  - los de tipo Color y Move usan imágenes cargadas desde el directorio del proyecto
    - los de tipo color solo tienen 3 imágenes correspondientes a los 3 primarios (R,G,B) - pendiente extenderla con mas colores
  - se está desarrollando una solución para que modifiquen los comandos lógicos del programa en tiempo de ejecución correctamente (una vez colocados en el Escritorio)

- Se ha refinado la creación de los programas lógicos mediante la colocación progresiva de comandos gráficos en el escritorio, respetando las reglas de la gramática desarrollada
  - se ha verificado que los comandos lógicos que forman los programas se añaden a estos en el orden correcto, mediante mensajes de debug lanzados con el botón Start del P.Control
  - se ha implementado correctamente el anidamiento de sentencias en los programas
  - queda definir la forma en que los comandos gráficos se deben eliminar para preservar la cohesión de los programas lógicos
- Se han asociado correctamente las acciones semánticas de los programas lógicos con las acciones reales que debe realizar el Robot en el Escenario
  - se ha probado que el robot responde correctamente a las acciones de movimiento (Move, SetPosition y SetSpeed)
  - pendiente refinarlo para poderse modificar en tiempo de ejecución (sobre los comandos gráficos ya colocados en el escritorio)
  - pendiente probar los comandos sensores y las acciones de movimiento dentro de sentencias de control
    - para esto es un prerequisito implementar los sensores del P.Control (última tecla y color actual)
- Se ha modificado la arquitectura cambiando la clase Comando para que actue como una clase abstracta, padre de los diferentes tipos de comandos gráficos (ComandoMove, ComandoColor ...)
- Se ha cambiado el look & feel de la GUI para que sea de tipo nimbus (<http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html>) y tenga un acabado mas agradable visualmente
- Pendiente de implementar la funcionalidad de anclar los comandos gráficos al Escritorio para no moverlos cuando ya formen parte de un programa
  - ver como afecta a la posibilidad de eliminarlos y/o modificar el programa
- Está pendiente el diseño gráfico de escenarios de fondo y robots variados en IllustrStudio / Photoshop (las pruebas de momento se hacen con imágenes tomadas de diferentes sitios web)

## 7.2.6. Plan de Iteración C2

### 7.2.6.1. Iteración Actual

- Construcción 2 (C2)

**7.2.6.2. Objetivos de Alto Nivel**

- requisitos terminados en su mayoría
- refinar DCD
- refinar Arquitectura SW
- implementación UC (75 %)
- pruebas UC (75 %)

### 7.2.6.3. Items de Trabajo

Cuadro 7.8: Items de trabajo iteración C2

| Descripción                                                                                                                      | Esfuerzo estimado | Área                   | Notas                        |
|----------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------|------------------------------|
| definir Plan de Iteración (C2)                                                                                                   | 1h                | G.Pr                   |                              |
| Refinar UC2<br>- lógica del dominio<br>- pruebas snap                                                                            | 10h               | Req, Dis,<br>Impl, Pru | Alta Prioridad (alto riesgo) |
| Panel de Control<br>- añadir ult. tecla / color<br>- relaciones con la GUI / robot                                               | 5h                | Req, Dis,<br>Impl, Pru | Baja Prioridad               |
| Refinar Lenguaje de Comandos<br>- análisis sintáctico<br>- acciones semánticas<br>- relaciones arquitectura<br>- patron Observer | 10h               | Req, Dis,<br>Impl      |                              |
| Refinar robot / fondo<br>- asociar robot con semántica<br>- pruebas (panel control)                                              | 20h               | Dis, Impl,<br>Pru      |                              |
| Refinar Arquitectura lógica<br>- relaciones GUI - Dominio<br>- reducir acoplamiento                                              | 4h                | Dis                    |                              |
| Modelo de Casos de Uso<br>- revisar modelo<br>- actualizar contratos                                                             | 2h                | Req                    |                              |
| Diseño Comandos<br>- diseño gráfico<br>- puntos de anclaje<br>- validar diseño                                                   | 5h                | Dis, Impl,<br>Pru      | Precondición UC2             |
| Refinar presentación<br>- layout del entorno<br>- look & feel                                                                    | 2h                | Req, Dis,<br>Impl      | Baja prioridad               |
| valoración iteración (C2)                                                                                                        | 1h                | G.Pr                   |                              |
| changelog y aspectos abiertos                                                                                                    | 2h                | Otros                  |                              |
| definir Plan de Iteración siguiente<br>(C3)                                                                                      | 1h                | G.Pr                   |                              |

#### 7.2.6.4. Problemas y Aspectos Abiertos

| Problema              | Estado        | Nota |
|-----------------------|---------------|------|
| GhostMoveManager      | pendiente     | 1    |
| planificación         | -             | 2    |
| creación de programas | en desarrollo | 3    |
| comboBox              | en desarrollo | 4    |
| hiperenlaces          | pendiente     | 5    |
| pruebas               | pendiente     | 6    |
| usabilidad            | -             | 7    |
| ultima tecla / color  | pendiente     | 8    |
| modelo UC             | refinar       | 9    |
| DI                    | pendiente     | 10   |

- 1 pendiente solucionar la accion de eliminar un comando lógico de un programa al mover el comando gráfico asociado por el escritorio
- 2 es probable que se añada una 4<sup>a</sup> iteración a la fase de Construcción, modificando el plan del proyecto en consecuencia. Se valorará la inclusión de iteraciones adicionales posteriores según el avance del proyecto
- 3 se necesita refinar la lógica de la creación de programas lógicos y la adición / eliminación de sus comandos lógicos constitutivos
- 4 pendiente de realizar mas pruebas que impliquen el uso del jComboBox en los comandos (uso de elementos de color para el Sensor Color, p.e)
- 5 pendiente de insertar hiperenlaces en la memoria (sobre todo UC's)
- 6 pendiente documentar resultados de las pruebas formalmente (dir pruebas)
- 7 se valora la posibilidad de dejar los mensajes de JOptionPane, que en principio estan destinados a debug, para informar a los alumnos de eventos como el snap (a falta de indicador visual mejor)
- 8 pendiente de incluir estos elementos en el Panel de Control
- 9 revisar mUC y contratos para incluir la lógica del dominio en los que sea necesario, como en el caso del CO5
- 10 pendiente completar los diagramas de interacción para los UC5, 6 y 7 (Panel de Control)

#### 7.2.6.5. Valoraciones

- Se han realizado un diseño gráfico preliminar de los comandos para orientar las acciones de snap y la gestión de la gramática

- Se ha modificado el diseño de los comandos gráficos para incluir una parte editable por los usuarios (JComboBox)
- Se ha completado parcialmente la acción de acoplar comandos (snap), de modo que responda gráficamente como se espera, y relacionando la lógica del dominio con las acciones gráficas
  - pendiente de refinar y hacer mas robusto el acoplamiento gráfico (el acoplamiento de los sensores sobre todo)
  - pendiente de refinar la lógica del dominio (añadir y quitar comandos lógicos de los programas, sobre todo quitarlos - pendiente en el GhostMoveManager)
- Se ha realizado una aproximación inicial para asociar las acciones semánticas con las acciones reales del robot
  - patron singleton: los comandos lógicos acceden a la unica instancia de robot para modificarla segun convenga
- Se han establecido las relaciones entre los botones del panel de control y las acciones de los programas del escritorio, mediante el uso de un patron observador
  - el Panel de Control publica eventos a los cuales se suscriben las clases que implementan la interfaz PropertyListener
  - tanto el Escenario (para cargar el robot y el fondo) como el Escritorio (para ejecutar, parar o resetear programas) implementan dicha interfaz, estableciendo sus propias acciones a los mensajes que publica el Panel de Control
- Se han actualizado los contratos de las operaciones para añadir la lógica del dominio en la operación colocar comando (CO5)
  - se ha incluido un DI llamado placeCommand(), en el área de diseño, para reflejar la mecánica de esta operacion a nivel lógico (dominio)
- Se ha modificado la vision de la arquitectura añadiendo el package «Ghost», encargado del acciones soporte del DnD, a la capa de servicios
- Se ha reorganizado la paleta de comandos para que muestre todos los comandos disponibles, agrupados por pestañas según la tipología de estos
- Pendiente añadir elementos sensores «última tecla» y «color» al panel de control
- Probablemente sea necesaria una iteración mas en la fase de Construcción (C4)
  - los % de consecución de implementación y pruebas no creo que estén al nivel reflejado en los requisitos, de modo que se modificará el plan general en consonancia
  - calculo que el % de impl / pruebas estará en torno al 60 % realmente, teniendo en cuenta los aspectos pendientes

**7.2.7. Plan de Iteración C1****7.2.7.1. Iteración Actual**

- Construcción 1 (C1)

**7.2.7.2. Objetivos de Alto Nivel**

- revisar Modelo UC (90 %)
- refinar DCD
- refinar Arquitectura SW
- implementación UC (50 %)
- pruebas UC (50 %)

### 7.2.7.3. Items de Trabajo

Cuadro 7.9: Items de trabajo iteración C1

| Descripción                                                                                                                                             | Esfuerzo estimado | Área                   | Notas                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------|------------------------------|
| definir Plan de Iteración (C1)                                                                                                                          | 1h                | G.Pr                   |                              |
| Refinar UC2<br>- revisar definición y diseño<br>- revisar DSS / DI<br>- revisar Implementación / pruebas<br>- documentarse sobre Java Shape             | +20h              | Req, Dis,<br>Impl, Pru | Alta Prioridad (alto riesgo) |
| Refinar UC3<br>- DSS y DI<br>- ver como afecta UC2 y su modificación                                                                                    | 2h                | Dis, Impl              |                              |
| Refinar Lenguaje de Comandos<br>- análisis sintáctico<br>- acciones semánticas                                                                          | 10h               | Req, Dis,<br>Impl      |                              |
| Diseño inicial robot / fondo<br>- DCD y arquitectura<br>- implementación<br>- pruebas                                                                   | 10h               | Dis, Impl,<br>Pru      |                              |
| Revisar Arquitectura lógica<br>- incluir package lenguaje<br>- organización en capas / paquetes<br>- relaciones GUI - Dominio<br>- reducir acoplamiento | 4h                | Dis                    |                              |
| Diseño e Implementación UC8 y 9<br>- elementos GUI responsables<br>- implementación<br>- pruebas                                                        | 5h                | Dis, Impl,<br>Pru      | Baja Prioridad (bajo riesgo) |
| valoración iteración (C1)                                                                                                                               | 1h                | G.Pr                   |                              |
| changelog y aspectos abiertos                                                                                                                           | 2h                | Otros                  |                              |
| definir Plan de Iteración siguiente (C2)                                                                                                                | 1h                | G.Pr                   |                              |

#### 7.2.7.4. Problemas

| Problema     | Estado        | Nota |
|--------------|---------------|------|
| UC2          | en desarrollo | 1    |
| DSS y DI     | completados   | 2    |
| UC1          | en desarrollo | 3    |
| Robot        | pendiente     | 4    |
| presentación | pendiente     | 5    |
| Observador   | pendiente     | 6    |
| Contratos    | pendiente     | 7    |

- Nota 1:
  - pendiente de definir los puntos de anclaje según tipo de comando.
  - Realizar mas pruebas sobre esta funcionalidad y ver por qué en ocasiones no reconoce el acoplamiento aunque el comando origen esté encima del comando destino.
  - Pendiente de diseñar las imágenes de los diferentes comandos para poder completar los puntos anteriores.
  - pendiente ver si añadimos un boolean «locked» a los comandos (gráficos) para impedir que se muevan una vez colocados, según que casos
  - verificar que la solución de los puntos de anclaje es viable
    - *otras posibilidades:* paquete shapes (shapeComponent, shapeIcon, método Graphics2D.fill(shape), etc..), clipping image with Shape, test de colisiones a partir de los píxeles de la imagen ....
- Nota 2: se han completado los diagramas de secuencia y de interacción de los UC0 - 3, sin embargo, aunque parecen bastante estables, es posible que se necesite modificarlos en proximas iteraciones en función de como avance el desarrollo
- Nota 3: al añadir un comando (gráfico) al escritorio, debemos además, añadir un comando lógico a la estructura sintáctica del programa objetivo (o crear uno si es Start). Pendiente
- Nota 4: si se presiona cualquier botón del panel de control, se bloquea la respuesta del robot de prueba a los eventos del teclado
- Nota 5: ajustar el layout del entorno y mejorar la presentación visual del mismo
- Nota 6: para la comunicación entre el panel de control y el escenario (en el UC8 y 9) se ha seguido el patron de diseño Observador; ver si esta misma solución es válida para la relación entre el escritorio (comandos lógicos) y el Robot / escenario
- Nota 7: pendiente de actualizar los contratos de las operaciones, y/o añadir nuevos en función de las necesidades

### 7.2.7.5. Valoraciones

- se ha implementado un prototipo de la acción de acoplar comando, mediante el solapamiento de las etiquetas de modo que las imágenes parezcan «acopladas». Es necesario ampliar y completar dicha solución, incluyendo puntos de anclaje para cada tipo de comando y añadiendo las restricciones de acoplamiento que impone la gramática desarrollada.
- se han actualizado los diagramas de secuencia y de interacción de los UC0 - 3
- se ha implementado una versión inicial del escenario, incluyendo el Robot y el Fondo, y añadiendo la posibilidad de mover el robot con las teclas de dirección. En próximas iteraciones se modificará el robot para que responda a las acciones semánticas indicadas por los programas del escritorio.
- los UC8 y 9 están implementados y probados, si bien es necesario ver si ampliamos su funcionamiento para incluir la posibilidad de cargar imágenes desde URLs
- se ha refinado la arquitectura sw, incluyendo el package lenguaje y las relaciones necesarias con dicho paquete. Pendiente refinar para reducir el acoplamiento con la GUI.
- pendiente de añadir los elementos «ultima tecla» y «color» al panel de control

## 7.2.8. Plan de Iteración E3

### 7.2.8.1. Iteración Actual

- Elaboración 3 (E3)

### 7.2.8.2. Objetivos de Alto Nivel

- definir UC detalle (40 %)
- completar Modelo de dominio
- refinar diseño de objetos, diagramas de clase
- refinar Arquitectura SW
- implementación UC adicionales
- probar UC adicionales

### 7.2.8.3. Items de Trabajo

Cuadro 7.10: Items de trabajo iteración E3

| Descripción                                                                                                                                             | Esfuerzo estimado | Área                         | Notas                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------|----------------------------------------------------------------------------|
| definir Plan de Iteración (E3)                                                                                                                          | 1h                | G.Pr                         |                                                                            |
| revisar Casos de Uso<br>- eliminar innecesarios (UC4)<br>- modificar existentes (UC2 y 3)<br>- añadir / completar nuevos (UC5)                          | 4h                | Req                          |                                                                            |
| Revisar UC2 - Acoplar comando<br>- revisar definición<br>- revisar DSS / DI<br>- revisar Implementación / pruebas                                       | +20h              | Req,<br>Dis,<br>Impl,<br>Pru | completar UC si es posible                                                 |
| Diagramas de Interacción UC2 y 3<br>- refinar (ing.inversa - ObjectAid)                                                                                 | 5h                | Dis                          |                                                                            |
| Definir clases de diseño de la capa de dominio                                                                                                          | +10h              | Dis / Impl                   | clases para definir el movimiento y las acciones del Robot en el Escenario |
| Definir UC5 Ejecutar programa (formato completo)<br>- Comenzar la definición del lenguaje de comandos<br>- Partir el UC5 en subcasos en fc del lenguaje | 10h               | Req                          | - alta prioridad                                                           |
| Refinar Arquitectura lógica<br>- organización en capas / paquetes<br>- relaciones GUI - Dominio<br>- reducir acoplamiento (ghost - gui)                 | 4h                | Dis / Impl                   |                                                                            |
| Definir UC 8 y 9 (f. completo)                                                                                                                          | 3h                | Req                          | - baja prioridad                                                           |
| valoración iteración (E3)                                                                                                                               | 1h                | G.Pr                         |                                                                            |
| changelog y aspectos abiertos                                                                                                                           | 2h                | Otros                        |                                                                            |
| definir Plan de Iteración siguiente (E4 o C1)                                                                                                           | 1h                | G.Pr                         |                                                                            |

### 7.2.8.4. Problemas

| Problema | Estado    | Nota |
|----------|-----------|------|
| UC2      | pendiente | 1    |
| UC5      | pendiente | 2    |
|          |           |      |

- nota 1: está pendiente continuar con este UC. Esto se realizará dentro de la fase de construcción, ya que aunque definido el funcionamiento a grandes rasgos, esta funcionalidad tiene aun mucho trabajo

- implementación del «análisis sintáctico» (comprobador de tipos)

- que comandos se pueden acoplar a cuales y como realizarlo (formas diversas?)
  - documentarse sobre etiquetas con formas libres (no rectangulares) - quizás clase Shape de JSwing
  - refinar verificación de colisiones
  - acción de acoplamiento de imágenes propiamente dicho
  - interacción con elementos del dominio de la aplicación (no solo a nivel de GUI)
- nota 2: refinar la definición del lenguaje
- relación del lenguaje con la GUI
  - acciones semánticas y su implementación

#### 7.2.8.5. Valoraciones

- Se ha realizado una definición inicial del lenguaje de comandos, especificando su léxico (tokens de comando), su sintaxis (gramática) y su semántica (acciones que representa cada comando)
  - queda refinar dicho lenguaje sobre todo a nivel de acciones semánticas
  - pendiente determinar como se realizará el análisis sintáctico, es decir, como establecer que construcciones de comandos en el escritorio se consideran válidas, y como conseguirlo a nivel operativo y a nivel de GUI
- Se han definido un conjunto de clases de diseño relativas al dominio de la aplicación, contenidas en el paquete «lenguaje», las cuales constituyen la implementación del lenguaje definido anteriormente
  - se ha creado asimismo un diagrama de clases de diseño (DCD) que muestra la arquitectura de dicho paquete
- Se ha completado el Modelo de Casos de Uso, de forma que todos los UC están escritos en formato completo, quedando la funcionalidad general del Sistema definida en anchura
  - Quedan pendientes de completar los UC10 y 11, relacionados con la gestión de usuarios y proyectos del Sistema; esta es una funcionalidad que me gustaría añadir al Sistema, pero de momento no la considero prioritaria, de modo que esto se postpondrá para mas adelante en función de como avance el proyecto
  - en principio la Fase de Elaboración la considero completa en este punto, ya que se han abordado los aspectos centrales del Sistema, y se han cubierto sus funcionalidades al menos en anchura, sin profundizar en sus soluciones definitivas, de modo que no se añadirá una 4<sup>a</sup> iteración de Elaboración (E4), iniciando en la siguiente iteración la fase de Construcción

- quizás la Fase de Construcción se amplie con una 4<sup>a</sup> iteración (C4), sin embargo esto se valorará mas adelante durante la propia fase
- se ha incluido el «UC0 Iniciar Entorno», para factorizar las operaciones comunes a los UC1, 2 y 3
- Se han incluido un fondo y un robot de prueba en el Entorno para ver las implicaciones que tiene esto a la hora de la creación inicial de la GUI (UC0 Iniciar Entorno)
  - estos son provisionales y estan incluidos solamente a nivel de pruebas
- Queda pendiente completar el UC2 acoplar comando
  - este UC es mas complejo de lo que parece a simple vista de modo que aún necesita bastante trabajo para hallar una solución a los problemas que plantea
- Pendiente refinar DI de UC2 (cuando se defina mejor este UC) y UC3
- Queda pendiente incluir en el diagrama de la Arquitectura lógica las relaciones especificadas en el DCD del lenguaje, asi como el paquete «lenguaje»

### 7.2.9. Plan de Iteración E2

#### 7.2.9.1. Iteración Actual

- Elaboración 2 (E2)

#### 7.2.9.2. Objetivos de Alto Nivel

- definir UC detalle (25 %)
- refinar Modelo de dominio
- refinar Modelo de diseño
- definir Arquitectura SW
- implementacion UC (25 %)
- probar UC (25 %)

### 7.2.9.3. Items de Trabajo

Cuadro 7.11: Items de trabajo iteración E2

| Descripción                                                                                                                    | Esfuerzo estimado | Área  | Notas |
|--------------------------------------------------------------------------------------------------------------------------------|-------------------|-------|-------|
| definir Plan de Iteración (E2)                                                                                                 | 1h                | G.Pr  |       |
| mUC: revisar Casos de Uso                                                                                                      | 1h                | Req   |       |
| mUC: DSS UC2 (flujo básico)                                                                                                    | 1h                | Req   |       |
| mUC: DSS UC3 (flujo básico)                                                                                                    |                   |       |       |
| refinar Modelo Dominio<br>(clases, asoc., atts)                                                                                | 2h                | M.Neg |       |
| mUC: Contratos UC2 y 3 (flujo básico)                                                                                          | 2h                | Req   |       |
| mDiseño: refinar realizacion UC1 (UCR1)<br>iniciar realizaciones UC2 y 3 (UCR2 y 3)<br>- Diag secuencia<br>- Diag colaboración | 4h                | Dis   |       |
| mDiseño: refinar DCD                                                                                                           | 3h                | Dis   |       |
| definir Arquitectura SW<br>(capas: 1-GUI, 2-Dom, 3-Support)                                                                    | 3h                | Dis   |       |
| mImplém: codificar clases del diseño                                                                                           | 3h                | Impl  |       |
| mImplém: implementar funcionalidades<br>completar UC1<br>iniciar UC2 y 3                                                       | 20h               | Impl  |       |
| mPruebas: Pruebas UC1, 2 y 3 (flujo basico)                                                                                    | 10h               | Pru   |       |
| valoración iteración (E2)                                                                                                      | 1h                | G.Pr  |       |
| changelog y aspectos abiertos                                                                                                  | 2h                | Otros |       |
| definir Plan de Iteración (E3)                                                                                                 | 1h                | G.Pr  |       |

### 7.2.9.4. Problemas

| Problema                 | Estado        | Nota |
|--------------------------|---------------|------|
| UC2 - Acoplar Comando    | en desarrollo | 1    |
| planificación            | -             | 2    |
| Modelo de Casos de Uso   | -             | 3    |
| Diagramas de Interacción | -             | 4    |

- *nota 1:* acoplar dos comandos en el escritorio no es una operación trivial, de modo que de momento se ha implementado una comprobación de solapamiento que informa de cuando se debería producir el snap
  - en ocasiones esta comprobación de solapamiento falla, cuando no debería. Pendiente de comprobar por que falla unas veces y otras no.
- *nota 2:* quizás se modifique la planificación general inicial, incluyendo una iteración E4 ( y quizás C4 ). Pendiente de valorar en la siguiente iteración

- *nota 3:* el modelo de casos de uso actual sufrirá modificaciones en la siguiente iteración, por el cambio de «Eliminar Comando» renombrado como «Mover Comando», y quizás la modificación del «UC2 - Unir dos Comandos». Asimismo el UC4 quizás se elimine, y se extienda el «UC5 - Ejecutar Programa» con subcasos de uso correspondientes a la semántica de cada Comando
- nota 4: los Diagramas de Interacción actuales no reflejan las operaciones a nivel de GUI de acoplar comando y eliminar comando. Se prevé obtenerlos mediante ingeniería inversa desde el código, mediante ObjectAidUML.
- *nota adicional:* se mantendrá la numeración de los casos de uso aunque se modifiquen, para facilitar la referencia a ellos, añadiéndose a continuación de los existentes en el caso de que surjan nuevos durante el desarrollo.

#### 7.2.9.5. Valoraciones

- Las funcionalidades de los Casos de Uso 1 y 3 están definidas, implementadas y probadas. Parecen bastante sólidas tal como está el sistema en este punto, no obstante, es posible que cambien en el futuro, mientras no se tenga una arquitectura definitiva del sistema. (por ejemplo, mediante la adición de un contenedor intermedio «Programa» para almacenar los comandos que se unan (UC2))
- La funcionalidad del UC2 está en desarrollo aún, quizás le falte mucho trabajo, ya que no parece una acción inmediata y trivial. Quizás se modifique la arquitectura para dar cabida a esta funcionalidad.
  - este UC se está desarrollando para JLabels ( o clases que la extiendan ) con forma RECTANGULAR. Esto se hace como aproximación inicial, pero es muy posible que los comandos tengan formas diferentes, para poder «encajar entre si», de modo que quizás se deba modificar esto. (clase Shape de Swing ?)
- Se ha definido una arquitectura lógica inicial del sistema. Es un diseño de alto nivel inicial que todavía debe ser refinado.
- Es necesario empezar cuanto antes con el UC5 y el diseño de las clases SW de los objetos del dominio (lógica de la aplicación)
  - quizás se priorice sobre la finalización del «UC2 - Acoplar Comandos», dejando este para la fase de Construcción

#### 7.2.10. Plan de Iteración E1

##### 7.2.10.1. Iteración Actual

- Elaboración 1 (E1)

### 7.2.10.2. Objetivos de Alto Nivel

- definir UC detallados en un 10 %
- definir un Modelo de Dominio inicial de la aplicación
- definir el Modelo de Diseño inicial con los diagramas de diseño necesarios
- comenzar la implementación del UC principal
- probar el sistema parcial resultante

### 7.2.10.3. Items de Trabajo

Cuadro 7.12: Items de trabajo iteración E1

| Descripción                                                | Esfuerzo estimado | Área  | Notas                                                |
|------------------------------------------------------------|-------------------|-------|------------------------------------------------------|
| mUC: DSS UC1 (flujo básico)                                | 0.5h              | Req   |                                                      |
| mDom: Diag UML<br>mDom: asociaciones<br>mDom: atributos    | 2h                | M.Neg |                                                      |
| mUC: Contratos UC1 (flujo basico)                          | 1h                | Req   |                                                      |
| mDiseño: Diag secuencia<br>mDiseño: Diag colaboración      | 4h                | Dis   |                                                      |
| mDiseño: realizacion UC1 (fb)<br>(UCR 1)                   | -                 | Dis   | diag sec<br>+ diag colab<br>+ DCD<br>aplicados a UC1 |
| mDiseño: DCD                                               | 3h                | Dis   |                                                      |
| mImplem: Código DCD                                        | 10h               | Impl  |                                                      |
| mPruebas: Pruebas UC1 (flujo basico)                       | 10h               | Pru   |                                                      |
| Arquitectura (capas: 1-GUI, 2-Dom, 3-Support)              | -                 | Dis   |                                                      |
| definir Plan de Iteración (E2)                             | 2h                | Req   |                                                      |
| completar Glosario (añadir)                                | 0.5h              | Req   |                                                      |
| Inicializacion entorno (GUI)<br>Creacion de ventanas Swing | 3h                | -     |                                                      |
| Creacion comandos de prueba (imgs)                         | 1h                | -     |                                                      |

- Para el UC1 debemos realizar la inicialización de las ventanas de la GUI (Swing)
  - al menos crear contenedores vacíos de prueba

#### 7.2.10.4. Problemas

| Problema                     | Estado    | Nota |
|------------------------------|-----------|------|
| Arquitectura (Gui / Dominio) | pendiente | 1    |
| Arrastrar comando (CO2)      | pendiente | 2,3  |
|                              |           |      |

- *nota 1:* determinar el diseño de la arquitectura, diferenciando que objetos son de la «capa GUI» y cuales de la «capa de dominio»
  - los objetos de la capa GUI se pueden implementar directamente en Swing (JComponents, etc...)
  - qué objetos serán implementados por Swing (GUI) y cuales no (dominio) ??
    - patrones : >>> Controlador (GUI recibe eventos y muestra info, y pasa msg al Controlador, que asigna responsabilidades al dominio)
  - implementamos un Swing por cada objeto de dominio susceptible de ello ? (capa GUI 1:1)
  - extendemos los paneles para que hereden de JPanel ( o JComponent ) ?
- *nota 2:* (UC1) CO2 - Arrastrar comando: entender mejor la lógica del DnD de Swing
  - Parece que reconoce las acciones de drag y drop, pero el panel dropTarget no importa los datos (no se copian en el escritorio)
  - Extensión: quedaría mejor a la hora de arrastrar que se mostrara el comando (la imagen) en el cursor en vez de el cursor típico (como en scratch)
- *nota 3:* (UC1) CO2 - Arrastrar comando: uso del ComponentMover
  - si utilizamos un solo JPanel para el panel de programación y arrastramos los comandos, éstos no se «copian» en el área del escritorio, desapareciendo de la paleta de comandos
    - podemos modificar el ComponentMover para añadir una funcionalidad de copiar, o
    - podemos trabajar con dos JPanels y usar el DnD de JSwing

#### 7.2.10.5. Valoraciones

- se ha hecho una primera aproximación al arrastre de comandos (funcionalidad (UC) principal de esta iteración), pero necesita refinarse bastante aún
- el desarrollo de la GUI probablemente necesite mucho mas esfuerzo del previsto inicialmente
- pendiente el desarrollo de la arquitectura (capas, separación GUI / Dominio)

## 8 Extensiones

En esta sección se indican todas las posibles extensiones y mejoras de funcionalidad previstas y pendientes de implementar:

- **Snap intermedio:** permitir el snap en medio de un programa y no solo al final del mismo (excepto Sensores que pueden acoplarse en cualquier orden)
- **Eliminar un solo comando:** habilitar una forma de eliminar comandos individualizadamente de un programa
- **Varios Robots:** valorar el uso de varios Robots, quizás usando varios Escritorios, uno por cada uno de ellos
- **Mapas Cerrados:** continuar ampliando la diversidad de los Mapas Cerrados, incluyendo:
  - Mas niveles de mapas
  - Items varios: trampas, portales, enemigos, etc
  - Historia en Mapas: posibilidad de desarrollar una historia del Robot que viaja a través de los mapas
  - Cambio de nivel al llegar al Portal: cargar el siguiente mapa de la historia al llegar al Portal
  - Items específicos para Sensor Color: crear algún tipo de objetos que definan el color de la casilla (orbes, globos, señales, etc.) para mejorar el uso del SensorColor
- **Cuentas de Usuario / Gestión de Proyectos:** valorar la posibilidad de ampliar la funcionalidad para poder crear sesiones de usuario y gestionar los proyectos por usuario

# 9 Bibliografía

## Tiles

- <http://opengameart.org/content/lots-of-free-2d-tiles-and-sprites-by-hyptosis>

## Especificaciones Iniciales

- Departamento de Lenguajes y Sistemas Informáticos de la ETS de Ingeniería Informática de la UNED

## Ghost

- <http://codeidol.com/java/swing/Drag-and-Drop/Translucent-Drag-and-Drop/>

## ColorComboBox

- <http://www.java2s.com/Code/Java/Swing-JFC/ColorComboBoxComboBoxEditorDemo.htm>
- <http://stackoverflow.com/questions/18830098/pick-color-with-jcombobox-java-swing>
- <http://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html>

## Unified Process

- <http://epf.eclipse.org/wikis/openup/>
- UML y Patrones, 2<sup>a</sup> Ed, Craig Larman

## Drag and Drop

- <http://docs.oracle.com/javase/tutorial/uiswing/dnd/intro.html>

## GUI

- <http://docs.oracle.com/javase/tutorial/uiswing/painting/index.html>

# 10 Anexo A: Manual de Usuario

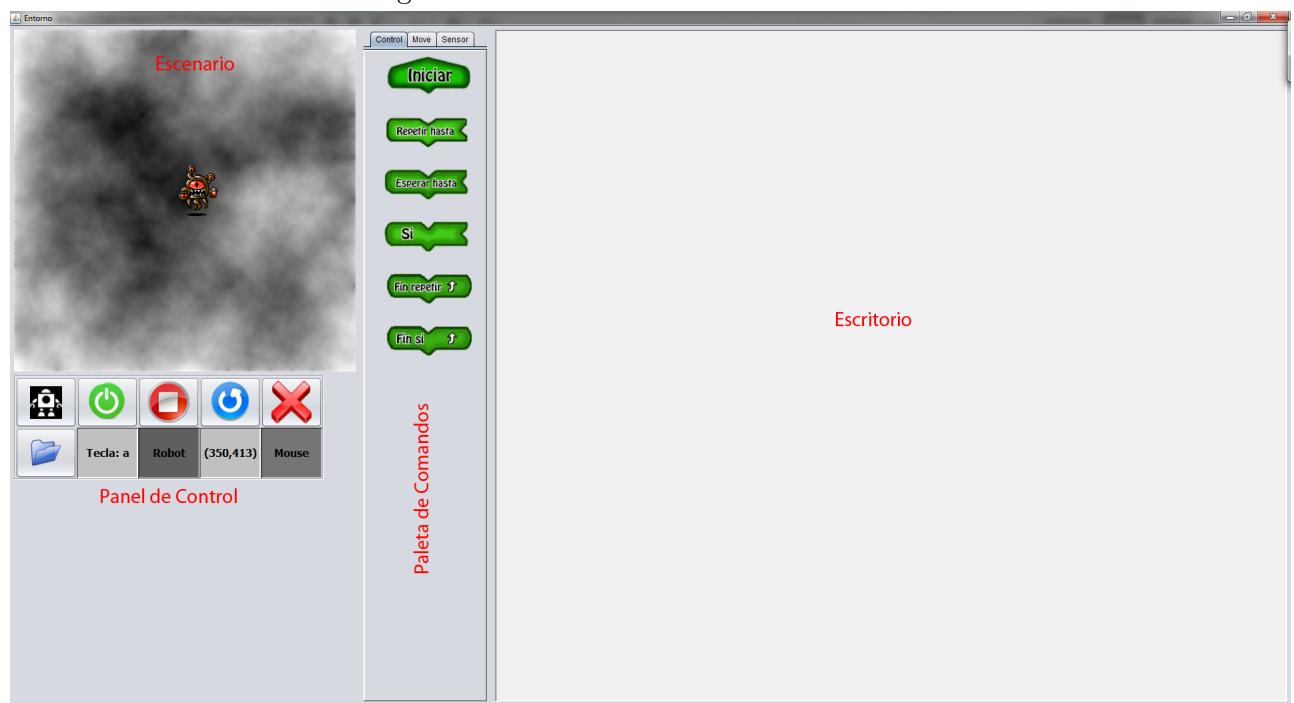
## 10.1. Instalación

Para la instalación de la aplicación, basta con descomprimir el archivo en cualquier directorio, siempre que el directorio de Imágenes esté en la misma ruta que el ejecutable.

## 10.2. Interfaz de Usuario

A continuación se muestra como es la interfaz de usuario, indicando cada uno de sus elementos:

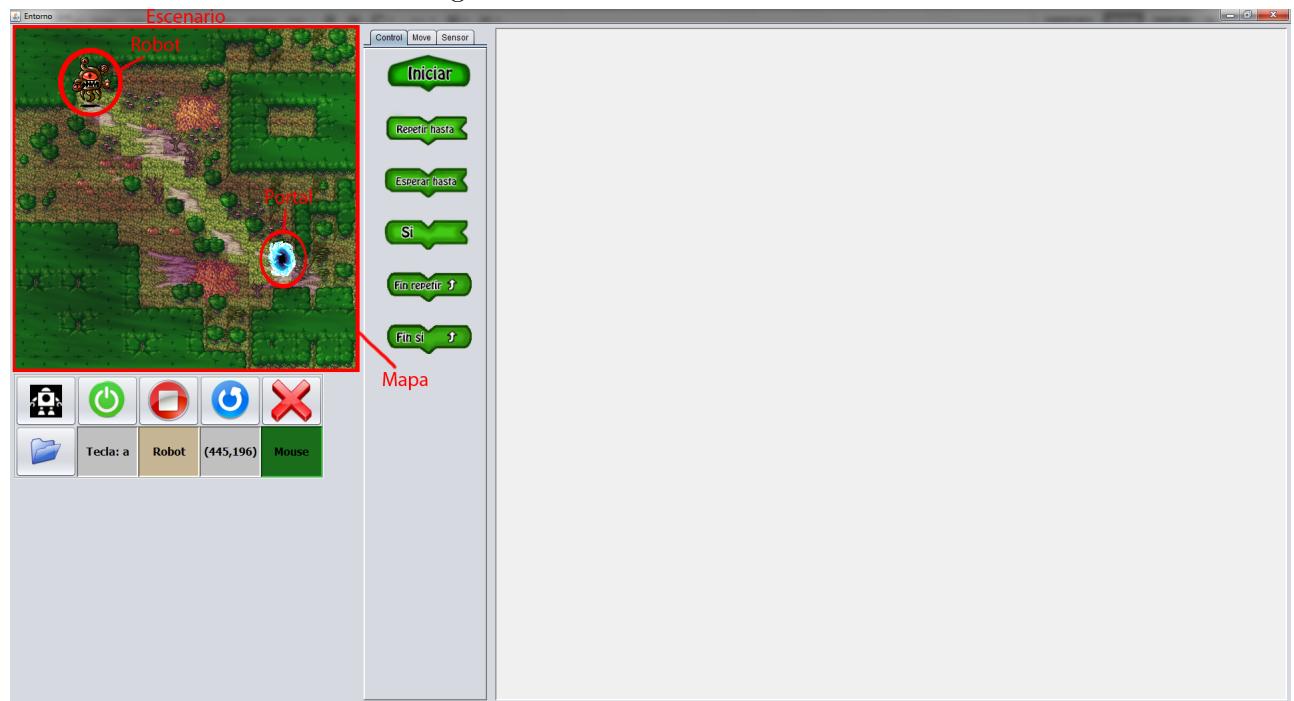
Figura 10.1: Iterfaz de Usuario



### 10.2.1. Escenario:

- panel en el cual se puede ver el movimiento del Robot en el Mapa actual

Figura 10.2: Escenario



- En la imagen de arriba se muestra el Escenario conteniendo un mapa cargado (en este caso es del tipo Mapas cerrados), el Robot y el Portal de salida (propio de este tipo de mapa)

### 10.2.2. Panel de Control:

- panel que contiene todos los botones e indicadores que sirven para controlar las acciones del Robot, y para indicar diferentes informaciones relacionadas con los programas. Los diferentes botones e indicadores se explican a continuación:

### 10.2.2.1. Botones:

Figura 10.3: Botones



**Cargar Robot:**

- carga una imagen desde una carpeta y la usa como imagen del Robot



**Cargar Mapa:**

- carga una imagen desde una carpeta y la usa como imagen del Mapa del Escritorio
- hay dos tipos de Mapas que se pueden usar, los cuales se comentan en su sección específica:
  - **Mapas cerrados**
  - **Mapas abiertos**



**Start:**

- al presionarlo inicia todos los programas presentes en el Escritorio

- solo funciona si los programas no están parados
- NOTA: no se debe pulsar varias veces seguidas, ya que en dicho caso se ejecutarán varias copias de los programas del Escritorio



- actúa como un interruptor con dos estados que detiene la ejecución de los programas e impide que se ejecuten mientras esté en modo stop, y reanuda y permite la ejecución en modo resume
  - *modo Resume*: en este modo (el inicial por defecto) se reanuda y permite la ejecución de los programas y muestra el ícono del siguiente estado (modo Stop) al que se pasará cuando se pulse de nuevo



- *modo Stop*: en este modo se detiene y prohíbe la ejecución de los programas y se muestra el ícono de estado siguiente (modo Resume)



**Reset:**

- solo funciona en el modo Stop
- si estamos en modo Stop, se posiciona al Robot en su posición inicial

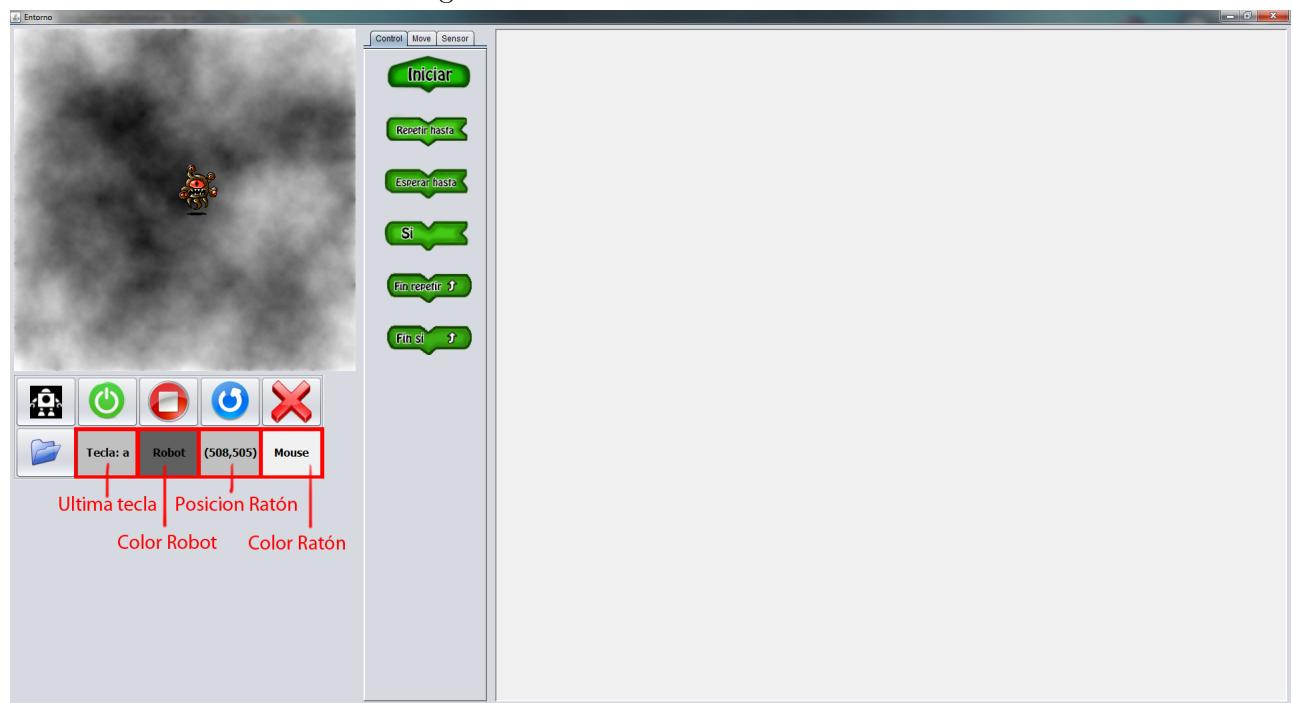


**Limpiar Escenario:**

- limpia el Escritorio, eliminando todos los Comandos Gráficos, así como los Programas Lógicos creados

### 10.2.2.2. Indicadores

Figura 10.4: Indicadores



#### Última Tecla

- muestra la última tecla presionada
- muestra correctamente caracteres, teclas numéricas, y algunos símbolos especiales
  - no muestra algunas otras teclas (no presentes en los comandos sensores de tecla): espacio, enter, etc..

#### Color del Robot

- muestra el color de la posición actual del Robot, obtenido del pixel correspondiente a la posición actual del robot (centrado en el tile)
- al iniciar el entorno el color actual que muestra es blanco (255,255,255) por defecto

#### Posición de Ratón

- muestra la POSICIÓN (coordenada (x,y)) en que se ha hecho click dentro del Escenario
- se usa para facilitar el comando Sensor de Color
- al iniciar el Entorno muestra un mensaje por defecto (coords)

### Color del Ratón

- similar al indicador anterior, sin embargo este muestra el COLOR de la coordenada del Escenario donde se ha hecho click por ultima vez
- se usa para facilitar el comando Sensor de Color
- al iniciar el Entorno muestra un color por defecto (blanco -> (255,255,255))

### 10.2.3. Paleta de Comandos:

- panel que contiene todos los Comandos del lenguaje de programación que se pueden usar para construir los programas que definen el comportamiento del Robot. Está dividido en 3 pestañas, cada una de las cuales contiene un tipo de Comandos

#### pestaña Control:

- contiene los Comandos de Control:

Figura 10.5: Paleta de Comandos



- Debe ser siempre el primer comando de un Programa (si no el programa no es válido y no se ejecutará)



- Repite las acciones de los comandos encerrados entre este comando y su comando asociado, hasta que se cumpla la condición de su Sensor asociado
- cada vez que se coloca un comando Repetir hasta, se abre un bloque de comandos interno que pertenece a este comando, el cual se cierra con un comando
- debe tener asociado un Comando Sensor, que define la condición de terminación del Repetir



- los sensores (condiciones) se pueden agregar en cualquier orden a los programas



- Hace que el programa se detenga hasta que se cumpla la condición definida por su comando Sensor asociado



- Las acciones encerradas entre este comando y su comando asociado se ejecutarán solo si se cumple la condición de su comando Sensor

- cada vez que se coloca un comando Repetir hasta, se abre un bloque de comandos interno que pertenece a este comando, el cual se cierra con un comando

- debe tener asociado un Comando Sensor, que define la condición de terminación del Repetir



- los sensores (condiciones) se pueden agregar en cualquier orden a los programas



- sirve para cerrar un bloque Repetir. Siempre cierra el bloque del comando Repetir inmediatamente anterior.

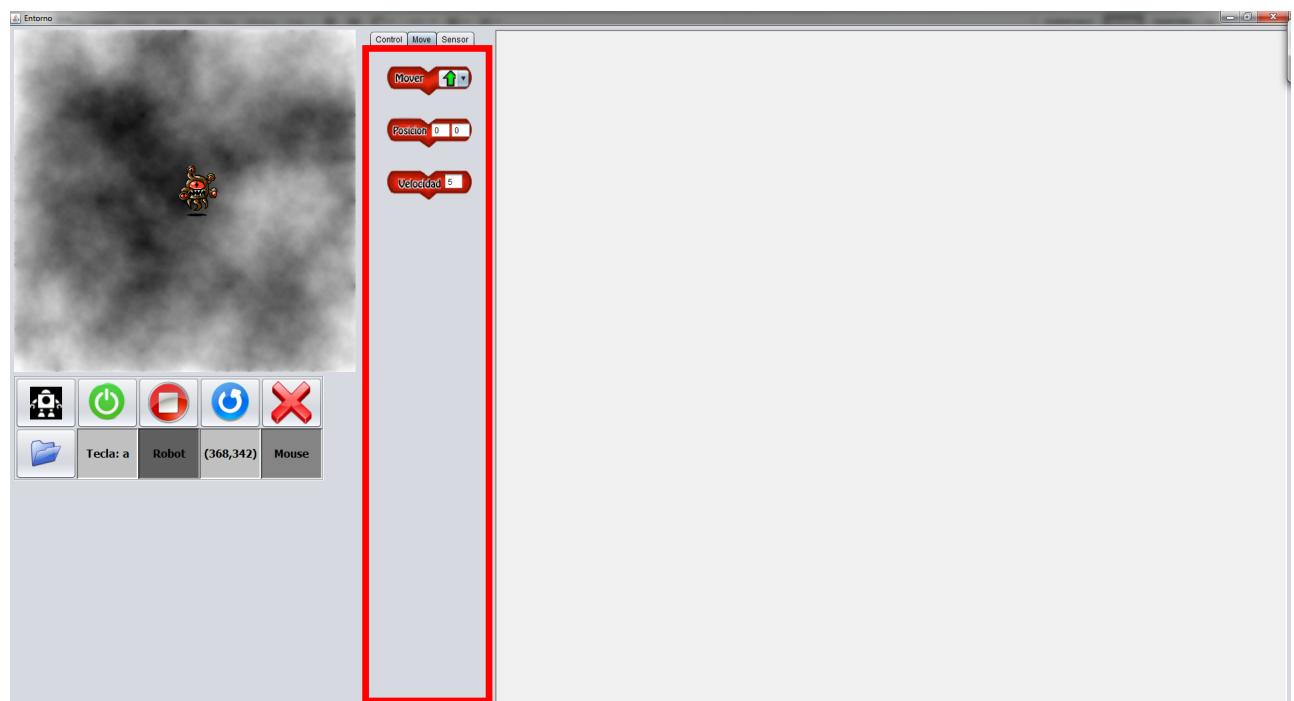


- sirve para cerrar un bloque Si. Siempre cierra el bloque del comando Si inmediatamente anterior.



#### pestaña Move:

- contiene los Comandos de Movimiento:



**Mover**

- mueve al Robot una posición en la dirección indicada por las flechas (arriba, abajo, derecha o izquierda)

**Posición**

- coloca al Robot en la posición indicada en las cajas de texto, siendo la primera caja correspondiente a la posición en x (ancho) y la segunda caja, la posición en y (alto)
- en el Escenario la esquina superior izquierda es la posición mas baja (0,0), y la esquina inferior derecha la posición mas alta (512,512)

**Velocidad**

- aumenta la velocidad a la que el Robot se mueve. Se pueden introducir valores entre 1 y 10, siendo 1 la velocidad mas baja y 10 la mas alta.
- por defecto la velocidad del Robot es 5.

**pestaña Sensor:**

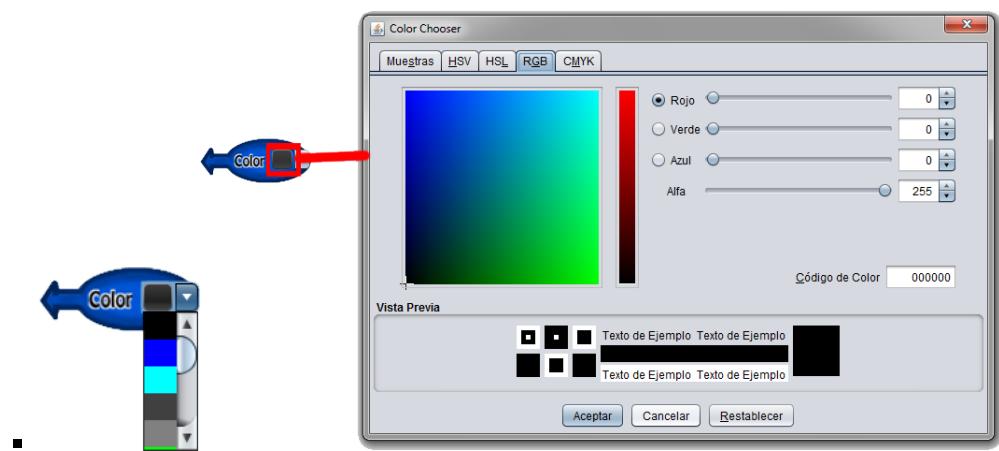
- contiene los Comandos Sensores:



- añade un Sensor de Color al comando de Control al que se une



- se puede elegir un color con la flecha desplegable, o bien se puede pinchar sobre la caja del comando y seleccionar cualquier color disponible.

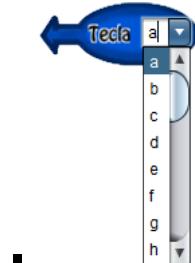




- añade un Sensor de Tecla al comando de Control al que se une



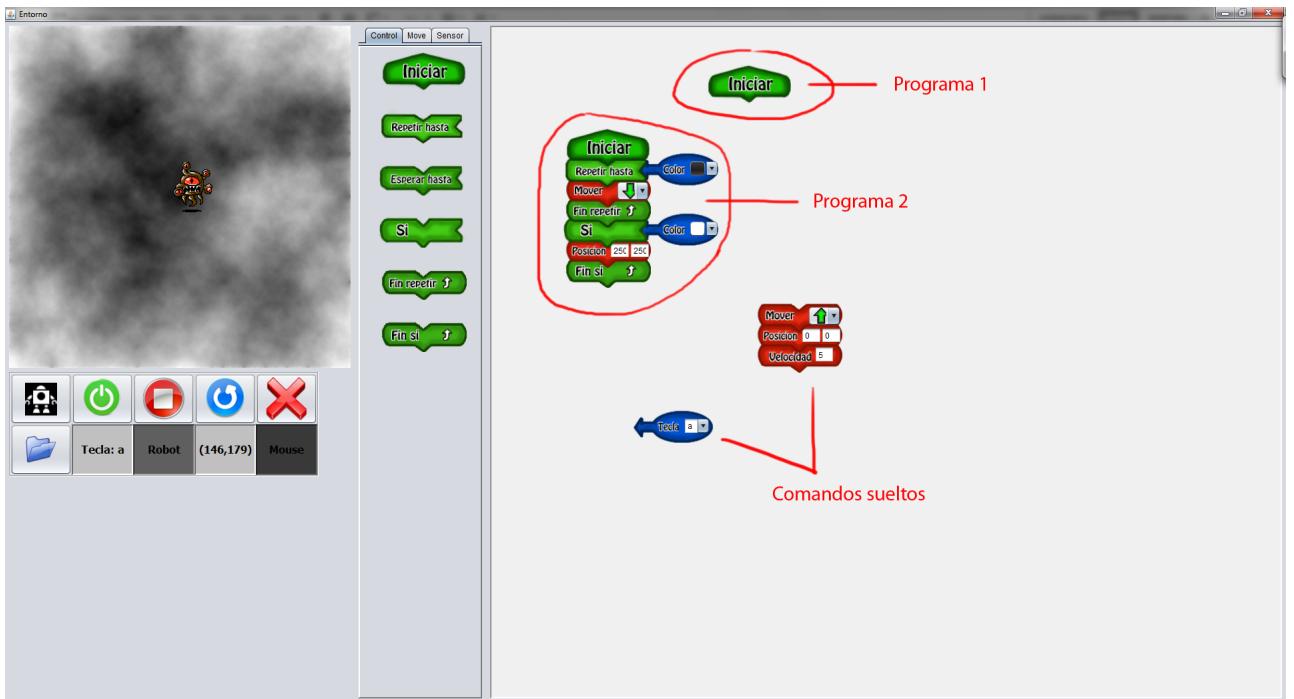
- se puede elegir una tecla con la flecha desplegable



#### 10.2.4. Escritorio

- Área donde se pueden colocar y unir los Comandos para formar los Programas que definen el comportamiento del Robot en el Mapa del Escenario
- En la siguiente captura podemos observar como se han colocado unos cuantos comandos en el Escritorio:
  - tal como están en la imagen, existen en el Escritorio 2 programas creados, uno vacío, que solo contiene el comando Iniciar, y otro con un bucle y un condicional a continuación
  - el resto de comandos libres no forman programas ya que no empiezan con el comando Iniciar

Figura 10.6: Escritorio



## 10.3. Mapas

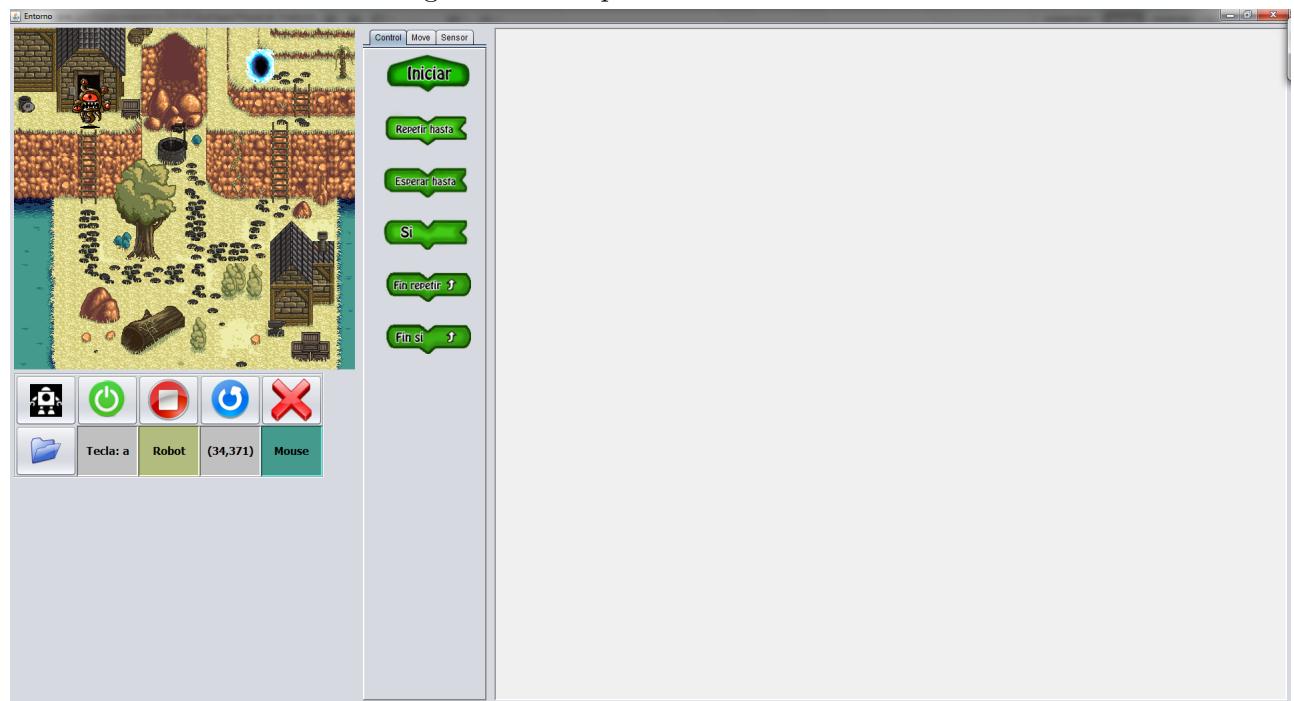
### 10.3.1. Mapas cerrados

- Son imágenes predefinidas para la aplicación, las cuales constan de dos imágenes asociadas:
  - mapa visible:** es una imagen de 512x512 px , creada con casillas o tiles de 32x32 px, con un camino que lleva de la salida a la meta, y en el que puede haber otras salidas, o trampas, u otros elementos
  - mapa negativo:** es el mapa asociado al mapa visible, a través del nombre (mismo nombre que el visible, pero con «Negative\_» como prefijo), el cual determina mediante colores cual es el camino válido (tiles en blanco), donde esta la salida, la meta, los diferentes objetos (si los hubiere) y el resto son tiles en negro (posiciones no válidas)

#### Ejemplo

- El siguiente sería un ejemplo del entorno usando un mapa cerrado

Figura 10.7: Mapa cerrado



- En este caso el Robot (representado por la animación del alien) tendría que recorrer el camino marcado por las piedras y escaleras para llegar desde la salida de la casa, hasta el portal en lo alto de la colina de la derecha.
- En este mapa, el mapa negativo nos define por debajo, cual es el camino válido a seguir (no-negro), donde está la salida (azul) y donde la meta (verde)
- Asimismo, también puede definir la posición de otro tipo de casillas (rojo), donde se podrían posicionar otros elementos, como trampas, objetos, enemigos, etc.

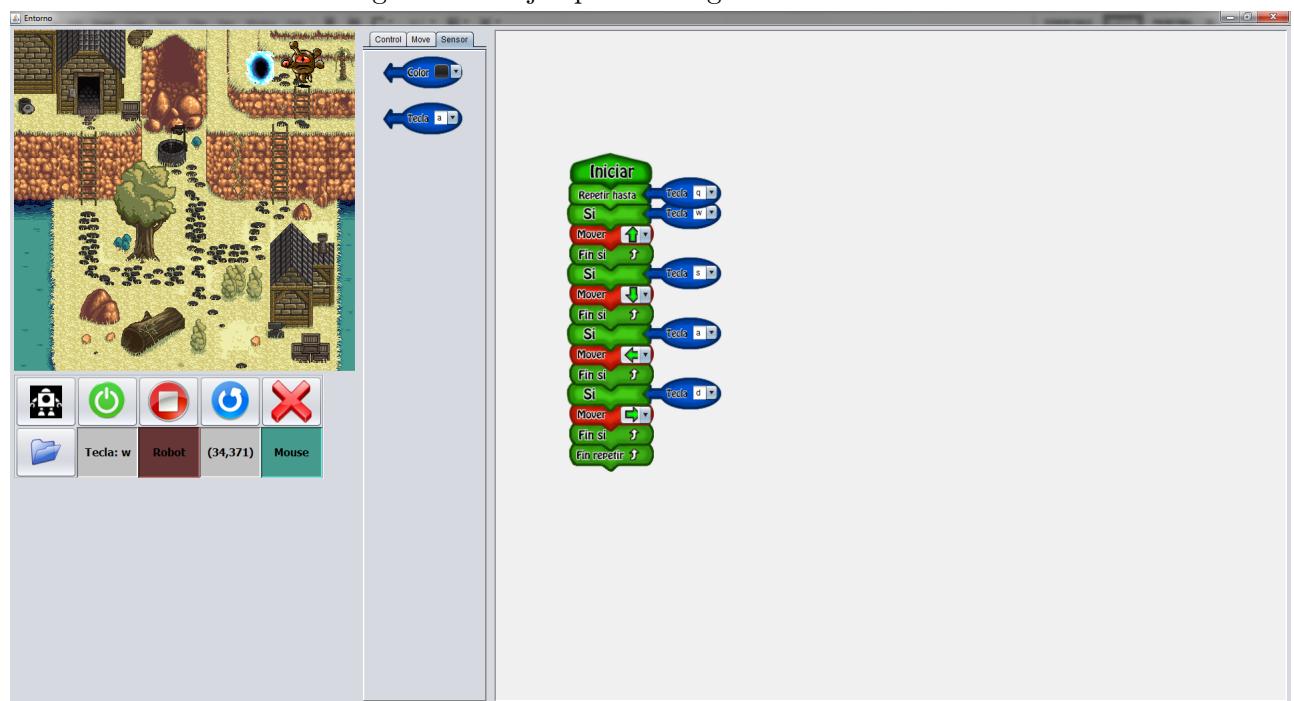
Figura 10.8: Mapa y Mapa Negativo



■

- A continuación se muestra un ejemplo de programa que se podría usar para recorrer el camino de este mapa y llegar a la meta

Figura 10.9: Ejemplo de Programa



- Como se puede ver, según dicho programa se deberían pulsar las teclas de dirección, w, s, a, d, en cada cambio de dirección del camino para dirigir al Robot hasta el portal. Sin embargo esta es solo una de las muchas posibles soluciones de programas que podrían crear los alumnos para resolver este Mapa.

### 10.3.2. Mapas abiertos

- Son el resto de imágenes que pueden ser cargadas, las cuales no tienen asociado un mapa negativo, sino que comparten un mapa Negative\_default, el cual es enteramente blanco, para dejar libertad total de movimiento (hasta los bordes)

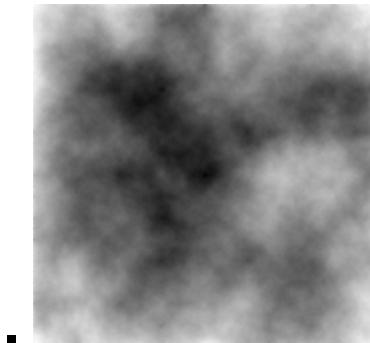
#### Ejemplo

- El siguiente es un ejemplo de mapa abierto cargado ya en el Entorno

Figura 10.10: Mapa Abierto



- Este tipo de mapas no tiene restricciones de camino, y se usan para dejar que los usuarios experimenten con el comportamiento del Robot creando diferentes programas, para probar las diferentes acciones que se pueden realizar
- Todos los mapas abiertos utilizan un mapa negativo por defecto, que es completamente blanco, para permitir libertad de movimiento
  - los mapas negativos por defecto se crean obteniendo las dimensiones de la imagen cargada, y creando otra a partir de esas dimensiones enteramente en blanco, de modo que se pueden usar imágenes de cualquier tamaño.
- Se puede cargar cualquier imagen como mapa visible, sin embargo esto influye en el tamaño del Escenario y por tanto en el tamaño del resto de paneles de la GUI, de modo que *se recomienda usar imágenes de tamaños no muy grandes ni muy pequeños*
- En caso de que no se pueda leer la imagen del archivo, se cargará un mapa en blanco de 512x512 px como mapa abierto visible (así como para el mapa negativo por defecto)



## 10.4. Créditos

### Desarrollo

- Octavio Martínez

### Diseño e Ilustraciones

- Octavio Martínez

### Tiles

- Mapas - Octavio Martínez
- TileSet - OpenGameArt (Hyptosis y Zabin)
- <http://opengameart.org/content/lots-of-free-2d-tiles-and-sprites-by-hyptosis>

### Especificaciones Iniciales

- Departamento de Lenguajes y Sistemas Informáticos de la ETS de Ingeniería Informática de la UNED

### Director del Proyecto

- Anselmo Peñas

### Agradecimientos

- Ghost
  - <http://codeidol.com/java/swing/Drag-and-Drop/Translucent-Drag-and-Drop/>
- ColorComboBox
  - <http://www.java2s.com/Code/Java/Swing-JFC/ColorComboBoxComboBoxEditorDemo.htm>

- <http://stackoverflow.com/questions/18830098/pick-color-with-jcombobox-javaswing>
- <http://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html>
- Unified Process
  - <http://epf.eclipse.org/wikis/openup/>
  - UML y Patrones, 2<sup>a</sup> Ed, Craig Larman
- Drag and Drop
  - <http://docs.oracle.com/javase/tutorial/uiswing/dnd/intro.html>
- GUI
  - <http://docs.oracle.com/javase/tutorial/uiswing/painting/index.html>
- Director PFG
  - Anselmo Peñas

## **11 Anexo B: Lista de Figuras**

A continuación se muestran los índices de figuras y cuadros utilizadas en la memoria

# Índice de figuras

|                                             |     |
|---------------------------------------------|-----|
| 1.1. Captura Scratch . . . . .              | 2   |
| 1.2. Captura Code.org . . . . .             | 3   |
| 3.1. Arquitectura SW - UML . . . . .        | 51  |
| 3.2. DCD paquete Ghost . . . . .            | 52  |
| 3.3. DCD paquete GUI . . . . .              | 53  |
| 3.4. DCD paquete lenguaje . . . . .         | 54  |
| 3.5. DI iniciarEntorno() . . . . .          | 55  |
| 3.6. DI elegirComando() . . . . .           | 55  |
| 3.7. DI arrastrarComando() . . . . .        | 56  |
| 3.8. DI snap/place() . . . . .              | 56  |
| 3.9. DI moverComando() . . . . .            | 57  |
| 3.10. DI ejecutarProgramas() . . . . .      | 57  |
| 3.11. DI detenerProgramas() . . . . .       | 57  |
| 3.12. DI resetProgramas() . . . . .         | 58  |
| 4.1. DI wait() . . . . .                    | 65  |
| 7.1. Muestra de artefactos del UP . . . . . | 92  |
| 10.1. Iterfaz de Usuario . . . . .          | 128 |
| 10.2. Escenario . . . . .                   | 129 |
| 10.3. Botones . . . . .                     | 130 |
| 10.4. Indicadores . . . . .                 | 133 |
| 10.5. Paleta de Comandos . . . . .          | 134 |
| 10.6. Escritorio . . . . .                  | 140 |
| 10.7. Mapa cerrado . . . . .                | 141 |
| 10.8. Mapa y Mapa Negativo . . . . .        | 141 |
| 10.9. Ejemplo de Programa . . . . .         | 142 |
| 10.10. Mapa Abierto . . . . .               | 143 |

# Índice de cuadros

|                                                |     |
|------------------------------------------------|-----|
| 2.1. Lista evento-actor-objetivo . . . . .     | 8   |
| 2.2. Glosario . . . . .                        | 46  |
| 2.3. Acrónimos del Proceso Unificado . . . . . | 46  |
| 7.1. Artefactos por áreas en el UP . . . . .   | 91  |
| 7.2. Calendario General del Proyecto . . . . . | 94  |
| 7.3. Items de trabajo iteración T2 . . . . .   | 96  |
| 7.4. Items de trabajo iteración T1 . . . . .   | 98  |
| 7.5. Items de trabajo iteración C5 . . . . .   | 101 |
| 7.6. Items de trabajo iteración C4 . . . . .   | 104 |
| 7.7. Items de trabajo iteración C3 . . . . .   | 108 |
| 7.8. Items de trabajo iteración C2 . . . . .   | 112 |
| 7.9. Items de trabajo iteración C1 . . . . .   | 116 |
| 7.10. Items de trabajo iteración E3 . . . . .  | 119 |
| 7.11. Items de trabajo iteración E2 . . . . .  | 122 |
| 7.12. Items de trabajo iteración E1 . . . . .  | 124 |

## **12 Anexo C: Palabras Clave para Búsquedas**

A continuación se detalla una lista de palabras clave sobre el contenido del proyecto con su traducción al inglés, a fin de facilitar su almacenamiento y recuperación en un sistema de búsqueda de bibliografía:

- Acción / Action
- Acoplar / Snap
- Botón / Button
- Bloque / Block
- Casilla / Tile
- Celda / Tile
- Comando / Command
- Comando gráfico / Graphical Command
- Comando lógico / Logical Command
- Envoltura / Wrapping
- Escenario / Scene
- Escritorio / Desktop
- Etiqueta / Label
- Fondo / Background
- Mapa / Map
- Paleta de comandos / Command Palette
- Papelera / Recycle Bin
- Programa / Program
- Robot / Robot

■ Unir / Snap

A continuación se incluye la traducción al inglés de la descripción indicada en Descripción para facilitar su almacenamiento y recuperación en un sistema de búsqueda de bibliografía.

- The project goal is to develop an educational application which will enable Primary School students to get used to programming and to develop their first computer programs. The application is a graphical environment, where students will be able to use sentences from a graphical programming language, consisting of preloaded images, to build their own programs. Those programs will control the behaviour of a robot, depicted by a loaded image or gif animation, that will move inside a scene, represented by a large background image loaded from file.