UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

MÁSTER UNIVERSITARIO
EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

Visualización dinámica mediante técnicas inmersivas para la optimización del proceso de aprendizaje de la programación de computadores

Cristian Gómez Portes

Febrero, 2020

VISUALIZACIÓN DINÁMICA MEDIANTE TÉCNICAS INMERSIVAS PARA LA OPTIMIZACIÓN DEL PROCESO DE APRENDIZAJE DE LA PROGRAMACIÓN DE COMPUTADORES

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de la Información

TRABAJO FIN DE MÁSTER

Visualización dinámica mediante técnicas inmersivas para la optimización del proceso de aprendizaje de la programación de computadores

Autor: Cristian Gómez Portes

Tutor: Dr. Javier Alonso Albusac Jiménez

Tutor: Dr. Carlos González Morcillo

Febrero, 2020

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de la Información

TRABAJO FIN DE MÁSTER

Visualización dinámica mediante técnicas inmersivas para la optimización del proceso de aprendizaje de la programación de computadores

Fdo.: Cristian Gómez Portes

Fdo.: Dr. Javier Alonso Albusac Jiménez

Fdo.: Dr. Carlos González Morcillo

Febrero, 2020

Cristian Gómez Portes

Ciudad Real – España

E-mail universidad: Cristian.Gomez2@alu.uclm.es
E-mail personal: Cristiancgph@gmail.com

© 2020 Cristian Gómez Portes

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

En la actualidad, la programación de computadores goza de un presente y futuro brillante, siendo uno de los estandartes que marca las líneas de progreso. De hecho, esta disciplina actualmente llega incluso a considerarse como la lengua del siglo XXI. Por un lado, cada año incrementa en Europa el número de puestos de trabajo que requieren habilidades de programación. Por otra parte, ésta se postula como una herramienta transversal que potencia el pensamiento computacional.

No obstante, uno de los mayores desafíos a los que se enfrenta la Ingeniería Informática es la enseñanza de la programación. Una de las principales causas de este problema es la abstracción que requiere esta disciplina y las serias dificultades que plantea a los estudiantes.

Esta problemática puede abordarse mediante la utilización de representaciones gráficas que capturen y mantengan la atención y la motivación del estudiante, asociando elementos del mundo real con términos específicos de programación. Además, el empleo de tecnologías inmersivas, como la Realidad Aumentada, puede mejorar el proceso de aprendizaje al introducir nuevos mecanismos innovadores de interacción y visualización.

Este nuevo modo de visualización puede permitir al estudiante estudiar códigos complejos con mayor grado de libertad, e incluso comprender mejor el flujo de ejecución en sistemas con alto grado de concurrencia donde existen diversos procesos e hilos actuando de forma simultánea.

El presente Trabajo Fin de Máster tiene como objetivo abordar el problema del aprendizaje de la programación mediante el uso de un sistema inmersivo basado en Realidad Aumentada. Para ello se propone una metáfora que soporta la visualización dinámica de programas, una arquitectura que añade el soporte computacional con el que se genera estas visualizaciones, y una evaluación correspondiente con estudiantes para demostrar la consecución de los objetivos.

Abstract

In today's society, computer programming enjoys a great present and future, being one of the standards that marks the lines of progress. In fact, this discipline now even comes to be considered as the language of the 21st century. On the one hand, the number of jobs that require programming skills is growing rapidly each year in Europe. On the other hand, programming is postulated as a transversal tool that helps to boost the computational thinking.

Nonetheless, one of the biggest challenges faced by Computer Science is the teaching of programming. One of the main factors of it is the abstraction that programming requires and the serious difficulties it causes for students.

This issue may be addressed through the use of graphic representations that capture and maintain the student's attention and motivation, associating real-world elements with specific programming terms. In addition, the use of immersive technologies, as Augmented Reality may improve the learning process, by introducing new innovative interaction and visualization mechanisms.

This new visualization mode may allow the student to study complex source code with a greater degree of freedom, and even better understand the flow of execution in systems with a high degree of concurrency where there are several processes and threads acting simultaneously.

This Master's Degree Final Project has the objective to address the programming learning problem through the use of immersive system based on Augmented Reality. To achieve this goal, this project proposes a metaphor that supports dynamic visualizations of programs, a new architecture that adds a computational support with which these visualizations are generated, and a corresponding evaluation with students to demonstrate the achievement of the objectives.

Agradecimientos

En primera instancia quisiera expresar mis agradecimientos a mis directores, Javier Alonso Albusac Jiménez y Carlos González Morcillo; gracias a ambos por la ayuda e implicación depositada en este trabajo.

Gracias a Miguel Ángel Redondo Duque por brindarme la oportunidad de participar en este precioso y ambicioso proyecto, aparte de proporcionarme los recursos necesarios para el desarrollo del mismo.

También quiero agradecer el apoyo recibido por mis dos compañeros y amigos de Máster, Carlos y Diego. Sin vosotros esta etapa no habría sido la mismo. Gracias a los dos.

Tampoco quiero olvidarme de mis compañeros del grupo de investigación CHICO. Agradecer a Yoel, Sergio, Nines, y por supuesto a Santi, a quien considero otro director de este trabajo. Muchas gracias por vuestros consejos, pero sobre todo por vuestra amistad.

A mis padres y a mi hermana por su apoyo incondicional; todo esto es gracias a vosotros.

Por último, pero no menos importante, a Alba, pues contigo no existen momentos duros. Gracias por confiar en mí.

Cristian Gómez Portes

Standing on the shoulders of giants

Índice general

Resumen V

Abstract VII

Agradecimientos IX

Índice general XIII

Índice de tablas XVII

Índice de figuras XIX

Índice de listados XXI

Listado de acrónimos XXIII

1. Introducción 1

1.1. Dificultades encontradas en el aprendizaje de la programación 2

1.2. Propuesta para facilitar el aprendizaje de la programación 3

1.3. Aprendizaje de la programación en el Trabajo Fin de Grado 4

1.4. Estructura del documento 5

2. Objetivos 7

2.1. Objetivo general 7

2.2. Objetivos específicos 7

2.2.1. Propuesta de una metáfora de carreteras 8

2.2.2. Diseño y codificación de un sistema de visualización dinámica . . . 8

2.2.3. Propuesta para la mejora de la programación concurrente 9

2.2.4. Evaluación y caso de estudio de la propuesta 9

3. Antecedentes 11

3.1. Visualización dinámica de programas 11

3.2. Efectividad de la visualización de programas 18
3.3. Realidad Aumentada 19
3.4. AsgAR y visualización estática 22
3.5. Conclusiones 26

4. Método de trabajo **27**
4.1. Propuesta para mejorar el proceso del aprendizaje de la programación . . . 27
4.1.1. Visión general de la propuesta 27
4.1.2. Metáfora de carreteras y señales de tráfico 29
4.1.3. Arquitectura del sistema 30
4.1.4. Sistema de depuración 33
4.1.5. Sistema de comunicación 37
4.1.6. Sistema de visualización dinámica 39
4.1.7. Sistema de gestión de archivos JSON 44
4.1.8. Sistema de efectos audiovisuales 46
4.2. Planificación 47
4.2.1. PT1. Propuesta de una metáfora de carreteras 48
4.2.2. PT2. Diseño y codificación de un sistema de visualización dinámica 48
4.2.3. PT3. Propuesta para la mejora de la programación concurrente . . . 51
4.2.4. PT4. Evaluación y caso de estudio para la programación concurrente 51
4.3. Metodología de trabajo 52
4.3.1. Desarrollo iterativo e incremental 52
4.3.2. Iteraciones 54
4.4. Medios empleados para el desarrollo 60
4.4.1. Medios *hardware* 60
4.4.2. Medios *software* 60

5. Resultados **67**
5.1. Visualización dinámica de programas 67
5.1.1. Preparación de la visualización 68
5.1.2. Visualización dinámica del algoritmo Búsqueda Lineal 70
5.2. Encuesta para la evaluación de la metáfora propuesta 70
5.2.1. Participantes 71
5.2.2. Equipamiento 71
5.2.3. Proceso 71
5.2.4. Análisis de los resultados 73

5.3. Conclusiones . . . 78
5.4. Análisis de costes . . . 79
5.5. Competencias adquiridas . . . 80
5.6. Publicaciones científicas . . . 80
5.7. Premios . . . 81

6. Propuesta de negocio **83**
6.1. Modelo de negocio de la propuesta . . . 83
6.2. Plan de trabajo . . . 85
6.3. Composición de equipo . . . 86
6.4. Retorno de inversión . . . 87

7. Conclusiones **89**
7.1. Objetivos alcanzados . . . 89
7.2. Líneas de trabajo futuras . . . 90
7.3. Conclusión personal . . . 91

A. Código fuente del proyecto **95**
A.1. Cliente . . . 95
A.2. Servidor . . . 95

B. Indicadores del curso 2018-2019 **97**

C. Modelo de encuesta para programación concurrente **99**
C.1. Pre-test . . . 99
C.2. Post-test . . . 99

D. Actividad a resolver por los alumnos **107**

E. GNU Free Documentation License **109**
E.0. PREAMBLE . . . 109
E.1. APPLICABILITY AND DEFINITIONS . . . 109
E.2. VERBATIM COPYING . . . 110
E.3. COPYING IN QUANTITY . . . 110
E.4. MODIFICATIONS . . . 111
E.5. COLLECTIONS OF DOCUMENTS . . . 112
E.6. AGGREGATION WITH INDEPENDENT WORKS . . . 112
E.7. TRANSLATION . . . 112

0.

E.8. TERMINATION 112
E.9. FUTURE REVISIONS OF THIS LICENSE 113
E.10. RELICENSING 113

Referencias **115**

Índice de tablas

4.1. Descripción resumida de la primera iteración 55
4.2. Descripción resumida de la segunda iteración 55
4.3. Descripción resumida de la tercera iteración 55
4.4. Descripción resumida de la cuarta iteración 56
4.5. Descripción resumida de la quinta iteración 56
4.6. Descripción resumida de la sexta iteración 56
4.7. Descripción resumida de la séptima iteración 57
4.8. Descripción resumida de la octava iteración 57
4.9. Descripción resumida de la novena iteración 58
4.10. Descripción resumida de la décima iteración 58
4.11. Descripción resumida de la undécima iteración 58
4.12. Descripción resumida de la duodécima iteración 59
4.13. Descripción resumida de la decimotercera iteración 59
4.14. Descripción resumida de la decimocuarta iteración 59

5.1. Resultado de las variables asociadas a la actitud del estudiante con respecto a la programación 74
5.2. Resultado de las variables asociadas a los conocimientos de programación . 74
5.3. Resultado de la comprensión e idoneidad de la metáfora de carreteras y señales de tráfico 74
5.4. Resultado de las variables asociadas a la motivación intrínseca y rendimiento subjetivo 77
5.5. Resultado de las variables asociadas a carga cognitiva de la tarea 77
5.6. Resultado de las variables que están asociadas a la percepción subjetiva de la calidad de la propuesta 78
5.7. Desglose de costes del proyecto 80

6.1. Plan de trabajo distribuido en 12 meses 86

Índice de figuras

1.1. Visión general del sistema 4

3.1. Interfaz de la herramienta VINCE 12
3.2. Visualización dinámica de algoritmo con Jeliot 3 13
3.3. Visualiación de algoritmo con SRec 14
3.4. Plataforma web diseñada para la enseñanza de la programación orientada a objetos 15
3.5. Ejemplo de utilización del entorno OGRE 16
3.6. Entorno RolePlay 17
3.7. Ejemplo del problema de los filósofos con el entorno IDEA 18
3.8. Taxonomía del concepto de RM de Milgran y Kishino 20
3.9. Ejemplos de marcas cuadradas y circulares 20
3.10. Notación ANGELA 23
3.11. Diseño de la interfaz gráfica del plug-in de visualización 24
3.12. Síntesis del procesamiento del código fuente de Java 25
3.13. Formato JSON relacionado con cada metáfora del conjunto de representaciones gráficas 26

4.1. Perspectiva global del sistema 28
4.2. Nuevo conjunto de representaciones gráficas que componen la notación ANGELA 30
4.3. Arquitectura del sistema 32
4.4. Elementos que componen el módulo de depuración 34
4.5. Interacción del usuario con el sistema para generar una visualización a partir de un fragmento de código 35
4.6. Proceso para desplegar el depurador de Eclipse 36
4.7. Perspectiva de depuración en Eclipse. Estructuras involucradas en el proceso de depuración. 37
4.8. Elementos que componen el módulo de visualización 40
4.9. Ejemplo de la generación de un grafo cíclico dirigido para el algoritmo la burbuja 43

4.10. Situación de un vehículo apunto de entrar en una sentencia condicional . . . 45
4.11. Intercambio de mensajes durante la visualización dinámica de un programa 46
4.12. Desglose de tareas y tiempo empleado 49
4.13. Propuesta de metáfora para el aprendizaje de la programación concurrente . 51
4.14. Modelo de desarrollo iterativo e incremental 53
4.15. Medios *software* 61

5.1. Distribución de imágenes que muestran la ejecución del algoritmo 69
5.2. Puntuación de comprensión para cada representación gráfica de la metáfora 75
5.3. Puntuación de idoneidad para cada representación gráfica de la metáfora . . 75
5.4. Porcentaje de alumnos que respondieron correctamente a cada una de las actividades propuestas 76

6.1. *Business Model Canvas* 84
6.2. Recuperación de inversión 87

Índice de listados

4.1. Estructura JSON para la gestión de la depuración 36
4.2. Método que gestiona la recepción de un mensaje del cliente 39
5.1. Algoritmo que cuenta las ocurrencias de un número que aparece de manera consecutiva en una lista 67
D.1. Pseudocódigo que representa el patrón de sincronización «barrera» 107

Listado de acrónimos

AST	Abstract Syntax Tree
API	Application Programming Interface
ANGELA	notAtioN of road siGns to facilitatE the Learning of progrAmming
CAD	Computer-Aided Design
CTIM	Ciencia, Tecnología, Ingeniería y Matemáticas
CLT	Cognitive Load Theory
CLR	Common Language Runtime
COM	Competencies
CPAS	Computer Programming Attitude Scale for University Students
CPKL	Computer Programming Knowledge Language
EFF	Effort
ESI	Escuela Superior de Informática
IU	Interfaz de Usuario
IDE	Integrated Development Environment
INT	Interest
ITU	Intention Of Use
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MRTK	MixedRealityToolkit
OSGI	Open Services Gateway initiative
PEOU	Perception Ease Of Use
PU	Perception Of Utility
PDE	Plug-in Development Environment
POO	Programación Orientado a Objetos
PSA	Perceived Subjective Attitude
PRE	Precision
RA	Realidad Aumentada
RM	Realidad Mixta
RV	Realidad Virtual
SWT	Standard Widget Toolkit
TAM	Technology Acceptance Model
TCP	Transmission Control Protocol

0.

TFG	Trabajo Fin de Grado
TFM	Trabajo Fin de Máster
TIC	Tecnologías de la Información y la Comunicación
UCLM	Universidad de Castilla-La Mancha
UDP	User Datagram Protocol
WMR	Windows Mixed Reality

Capítulo 1

Introducción

En la sociedad actual vivimos en una sociedad en la que convivimos con tecnologías que cooperan en nuestro desarrollo diario. Esto está dando lugar a que algunos sectores, tanto productivos como de servicios, se estén viendo afectados por la irrupción de las nuevas tecnologías. El sector educativo también está siendo influido por tales cambios, no solo con lograr mejores resultados en los alumnos, sino en formarlos en la nueva era digital [LL+15].

Este reto pasa porque los jóvenes adquieran las competencias suficientes para enfrentarles al nuevo mercado laboral, dotándoles de herramientas cognitivas que les ayuden a desenvolverse en esta dirección, haciendo uso del pensamiento computacional como paradigma de trabajo, y la programación como herramienta para la resolución de problemas [LGPMV17].

La idea del pensamiento computacional trata que desde la niñez se trabajen contenidos de diversas materias con enfoques informáticos, a fin de reforzar conceptos y desarrollando nuevos modelos mentales de más alto nivel [Win06]. Ante esta reflexión surge el reto de usar el pensamiento computacional como herramienta útil y eficaz en las áreas de conocimiento de Ciencia, Tecnología, Ingeniería y Matemáticas (CTIM)[1] [GPM18], usando la programación no solo como vehículo para reforzar este planteamiento, sino como conceptos de otros campos [Win08].

Esta situación junto a otras están convirtiendo la programación en una de las disciplinas más importantes de la actualidad. De hecho, existe un incremento en la actualidad del número de ofertas de trabajo que requieren de habilidades y de competencias digitales [Dis16]. Según calcula la Comisión Europea, en el año 2020 existirán alrededor de 900 000 puestos vacantes en el ámbito de las Tecnologías de la Información y la Comunicación (TIC) que necesitarán ser cubiertos en Europa [Ans15]. A modo de ejemplo, según el informe laboral elaborado por el Ministerio de Ciencia, Innovación y Universidades de España [min19], el 84,6 % de de los estudiantes que terminaron Informática en el curso 2013-2014 estaban afiliados a la Seguridad Social en 2018, cuatro años después de terminar sus estudios. En esta línea, Niel Kroes, vicepresidente de la Comisión Europea y responsable de la Agenda Digital para Europa, expuso en una carta conjunta enviada a los ministros de Educación de la Unión

[1] Equivalente en español de STEM: https://es.wikipedia.org/wiki/Educaci%C3%B3n_STEM

Europea[2] que la «programación es parte de la solución al desempleo juvenil en Europa».

Estos hechos ponen de manifiesto la importancia de la programación en la actualidad, al ser un campo que se postula en el ámbito académico como disciplina para trabajar contenidos de otras materias y, en el contexto profesional, para formar a nuevos profesionales y cubrir así la demanda actual de programadores.

1.1 Dificultades encontradas en el aprendizaje de la programación

Actualmente existe un problema con en el número de graduados en disciplinas que requieren conocimientos de programación. Estudios como en [KM06] constatan que el abandono de los estudiantes en programas de grado en Ciencias de la Computación está entorno al 40 % debido a la falta de motivación y tiempo que suponen asignaturas como la programación. A modo de ejemplo, la Escuela Superior de Informática de Ciudad Real presenta una tasa de abandono del 33,59 % en el curso 2014-2015. Es decir, el 33,59 % de los estudiantes de nuevo ingreso en el curso 2014-2015 abandonaron el Grado en el transcurso de los dos cursos posteriores a la matrícula (ver B). Existe, por tanto, una necesidad de retener a los futuros profesionales que se necesitan para hacer frente a la demanda actual de programadores.

Entre las principales causas del abandono de los alumnos se encuentra la abstracción que requiere la programación y las serias dificultades que plantea en los estudiantes, tales como la falta de conocimiento conceptual, uso incorrecto de estructuras sintácticas o la falta de creatividad, entre otras [SS12, DLRTH11, PC13].

En esta misma línea, otro de los problemas encontrados en la literatura sobre el aprendizaje de la programación está relacionado con la carencia que muchos estudiantes muestran a la hora de resolver problemas. Esta situación implica una dificultad en el diseño y creación de algoritmos, lo cual está supeditado a la habilidad general de abordar problemas [PC13].

Sin embargo, es importante destacar la complejidad que los docentes encuentran a la hora de explicar conceptos de programación. Gran parte de este problema reside en que los contenidos que se explican requieren de altas capacidades cognitivas por parte de los estudiantes que aún no poseen. Este proceso puede mejorarse mediante el empleo de abstracciones que faciliten la comprensión de estos conceptos. Sin embargo, el esfuerzo y tiempo por parte de los docentes para crear representaciones visuales ha contribuido a frenar este enfoque [HDS02].

Aparte de lo anterior, mantener la atención y aumentar el interés del estudiante durante el proceso de aprendizaje es otra de las dificultades a las que se enfrentan hoy. Esta situación se intenta evitar mediante clases en laboratorios donde se realizan actividades prácticas. Sin embargo, las actividades desarrolladas en estos espacios no son del todo efectivas por la falta de materiales y la dificultad de proporcionar ejemplos representativos [DA12].

[2] `http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=6597`

El presente Trabajo Fin de Máster (TFM) pretende dar una solución al problema del aprendizaje de la programación mediante el uso de un sistema inmersivo basado en Realidad Aumentada (RA). Esta propuesta presenta una metáfora, basada en carreteras y señales de tráfico, que soporta la visualización dinámica de programas y que da pie a utilizarse como medio para representar conceptos relacionados con la programación concurrente y en tiempo real. El trabajo incluye el desarrollo de una arquitectura que añade un soporte computacional con el que generar estas visualizaciones, junto a una evaluación con estudiantes del Grado en Ingeniería Informática para demostrar la utilidad e idoneidad de la metáfora.

1.2 Propuesta para facilitar el aprendizaje de la programación

En este trabajo se propone como objetivo facilitar el aprendizaje de la programación a estudiantes de grado que están aprendiendo a programar, quienes carecen de conocimientos suficientes para resolver problemas mediante un lenguaje de programación, concretamente Java.

Para ello, se propone un depurador interactivo en RA apoyado sobre un *plug-in* en el *Integrated Development Environment (*IDE*)* de Eclipse. Con esto se pretende reducir la complejidad ligada a la tarea de programar, visualizando dinámicamente programas con una metáfora que establece una analogía entre conceptos de circulación vial y aspectos de la programación. En este trabajo, esta metáfora es ampliada, añadiendo un cono de tráfico para representar el concepto de punto de ruptura y un vehículo con el que representar el concepto de hilo o proceso de ejecución. Dicha analogía define la notación *notAtioN of road siGns to facilitatE the Learning of prograMming (*ANGELA*)*, con la cual se representa la estructura y sentencias que componen un programa, junto a un vehículo que simula circular por dicha estructura ejecutando sentencias. Los alumnos que empiezan estudios superiores relacionados con la programación suelen ser estudiantes que acaban de alcanzar la mayoría de edad y que están en situación de obtener la licencia de conducir. De esta forma, pueden encontrar motivadora la metáfora propuesta al incluirse elementos de tráfico en ella. Además, este planteamiento es acertado para utilizarse como medio para representar conceptos de la programación concurrente, pudiendo simular mediante vehículos los procesos o hilos en ejecución de un programa. Dicho enfoque puede facilitar la comprensión de fragmentos de código basados en este nuevo paradigma de programación, el cual supone la gestión adecuada de las diferentes entidades que existen en ejecución. En la Figura 1.1 se presenta una panorámica general de la la ejecución del sistema que se propone.

Este TFM tiene como base el trabajo llevado a cabo en [GP18], el cual presenta ciertas debilidades y líneas de trabajo futuras. Las ampliaciones y mejoras incluidas en este trabajo vienen en parte gracias a los conocimientos adquiridos durante el propio Máster, como consecuencia de cursar, en concreto, la asignatura de Interacción y Visualización de la Información.

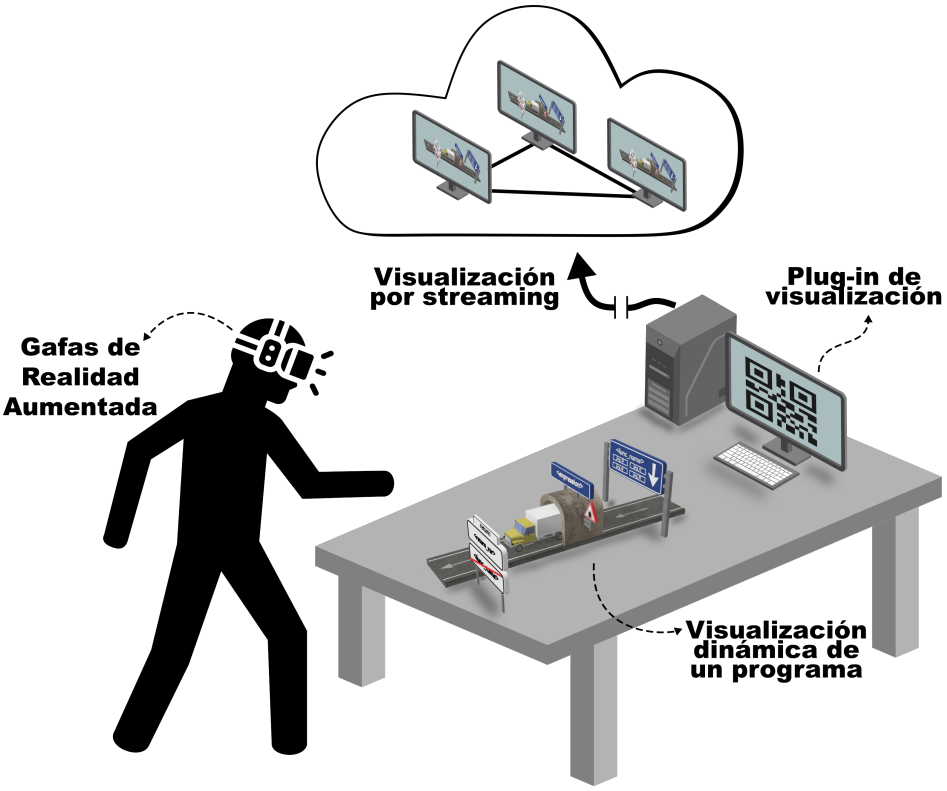


Figura 1.1: Visión general del sistema

Atendiendo a las consideraciones anteriormente mencionadas, este trabajo ofrece diferentes ventajas que facilitan el proceso de aprendizaje. Por un lado, el sistema permite crear escenarios de aprendizaje de aspectos de la programación, donde la utilización de esta notación facilita la realización de distintas tareas de aprendizaje. Un ejemplo de este escenario sería una lección en la cual un profesor visualiza la ejecución de un programa. En este caso, el profesor puede detener la ejecución durante la visualización y realizar preguntas acerca de lo que está ocurriendo, cambiando la dinámica de una clase convencional. Por otra parte, este sistema también facilita la generación de representaciones gráficas a partir de un soporte computacional desarrollado como un *plug-in* en Eclipse. Esta característica reduce el esfuerzo y tiempo por parte de los docentes en crear este tipo de representaciones que ayuden al aprendizaje de la programación.

1.3 Aprendizaje de la programación en el Trabajo Fin de Grado

El Trabajo Fin de Grado (TFG) presentado en [GP18] describe un sistema en RA que genera representaciones gráficas para construir visualizaciones estáticas basadas en la metáfora de carreteras, señales de tráfico y circulación vial. Dichas visualizaciones son programas ubicados y alineados sobre superficies del espacio físico que muestran su aspecto estático, es decir, estructuras de control y la lógica asociada que los componen.

La realización de este trabajo se debe a que existe un reto que gira en torno al problema del aprendizaje de la programación. Uno de los enfoques planteado en la literatura para resolver

esta problemática se basa en la utilización de representaciones gráficas para reducir el nivel de abstracción que requiere la programación en los estudiantes.

No obstante, el trabajo citado presenta algunas debilidades en cuanto a la mejora en el proceso del aprendizaje de la programación. Bien es cierto que la metáfora ayuda en la comprensión de conceptos de programación, al utilizar elementos cotidianos de la vida real. Sin embargo, el planteamiento diseñado es insuficiente, dado que solo permite representar programas o algoritmos en un modo estático, es decir, la estructura de la que se componen. Por lo tanto, este sistema presentaba ciertas carencias en cuanto a la mejora del proceso de aprendizaje, dado que no se mostraba la traza de ejecución de un programa lo cual permite comprender su funcionamiento. Incluso, la carencia anterior impide que se contemple la posibilidad de mostrar varios procesos e hilos en un mismo programa simultáneamente. Este enfoque se considera como uno de los más interesantes por el hecho de que la tecnología utilizada permite tener una visión global de todo los procesos en ejecución. Además, esto permite hacer un seguimiento personalizado de cada una de estas entidades al brindar libertad de movimiento. Este nuevo planteamiento permitiría comprender términos y mecanismos asociados a la programación concurrente y en tiempo real, dado que los depuradores que existen hoy en día para analizar el código ejecutado por varios hilos y procesos no son intuitivos y no facilitan el seguimiento de la ejecución.

1.4 Estructura del documento

El presente documento se ha estructurado según las indicaciones de la normativa de Trabajos Fin de Máster de la Escuela Superior de Informática (ESI) de la Universidad de Castilla-La Mancha (UCLM), empleando los siguientes capítulos:

Capítulo 1: Introducción

En este capítulo se presenta la idea general en torno a la cual gira este trabajo, exponiendo y describiendo una serie de conceptos que definen el tema tratado.

Capítulo 2: Objetivos

En este capítulo se desglosan y describen los objetivos y subobjetivos planteados que se han seguido para el desarrollo del presente trabajo.

Capítulo 3: Antecedentes

Este capítulo realiza una revisión en torno a los temas que trata este trabajo, recopilando las tecnologías y conceptos teóricos estudiados para el diseño e implementación del sistema.

Capítulo 4: Método de trabajo

Este capítulo describe cómo se ha diseñado e implementado el sistema. Además, se explica la metodología de trabajo empleada así como los diferentes materiales utilizados.

Capítulo 5: Resultados

Este capítulo refleja el resultado final obtenido tras lograr los objetivos propuestos al comienzo del proyecto.

Capítulo 6: Propuesta de negocio

En este capítulo se propone un posible modelo de negocio sobre la explotación de una evolución de este trabajo, cuyo plan es detallado con los potenciales paquetes de trabajo que se deberían materializar.

Capítulo 7: Conclusiones

Este capítulo resume los resultados más importantes logrados a lo largo de las fases de diseño, implementación y pruebas, así como posibles mejoras o evoluciones que pueden nacer a partir de este trabajo.

Capítulo A: Código fuente del proyecto

Este anexo muestra la estructura de directorios del código fuente relativo a este proyecto. Además se proporciona un enlace para su posible anáisis o descarga.

Capítulo B: Indicadores del curso 2018-2019

Este anexo muestra el conjunto de indicadores del curso 2018-2019 del Grado y Máster que se imparten en la Escuela Superior de Informática de Ciudad Real.

Capítulo C: Modelo de encuesta para programación concurrente

En este anexo se muestra la encuesta realizada a los estudiantes de la asignatura de Sistemas Operativos II del Grado de Ingeniería Informática para evaluar la notación ANGELA en el contexto de la programación concurrente.

Capítulo D: Actividad a resolver por los alumnos

Este anexo presenta el resultado de la actividad propuesta en la encuesta para evaluar la metáfora en el contexto de la programación concurrente.

Capítulo 2

Objetivos

Dado el enfoque y las ideas presentadas en la introducción, este capítulo se centra en detallar lo que se pretende llevar a cabo con este trabajo, sintetizando el objetivo general del proyecto y desglosando los objetivos específicos derivados.

2.1 Objetivo general

El objetivo principal del presente TFM es **diseñar e implementar un sistema de visualización dinámica de programas mediante RA para facilitar y mejorar el proceso de aprendizaje de la programación**. Este enfoque, con la ayuda de un dispositivo de RA, visualiza la traza y código fuente de un programa usando representaciones gráficas generadas de forma dinámica que asocian términos de circulación vial con conceptos específicos de programación, y que se ubican y alinean con cualquier superficie del mundo real. Además, este planteamiento tiene otra singularidad, y es que permite comprender conceptos y mecanismos usados en contextos donde existe concurrencia al visualizar y animar el comportamiento de estos elementos. Cabe destacar como característica diferenciadora el empleo de tecnología de visualización, la cual ayuda a mejorar el realismo de la analogía con los elementos físicos que representa.

En definitiva, este trabajo es una propuesta que ayuda en el aprendizaje de la programación para aquellos que no tienen experiencia en la tarea de programar. También ayuda a estudiantes con este perfil a introducirse en el mundo de la programación concurrente, aparte de ayudar a usuarios con otro perfil más experto a comprender el comportamiento de este tipo de programas, siempre utilizando la RA como tecnología para mejorar el proceso de aprendizaje.

2.2 Objetivos específicos

A partir del objetivo principal descrito en la sección anterior, este punto presenta los subobjetivos específicos que se pretenden abordar. Éstos pueden agruparse en torno a las siguientes líneas: (1) propuesta de una metáfora de carreteras y señales de tráfico para soportar la visualización dinámica de programas, (2) diseño y codificación de un sistema de visualización dinámica y (3) evaluación y caso de estudio para la programación concurrente.

2.2.1 Propuesta de una metáfora de carreteras y señales de tráfico para soportar la visualización dinámica de programas

En este objetivo se propone la modificación y ampliación de una metáfora elaborada en un TFG previo para habilitar la generación automática y dinámica de visualizaciones, así como la representación de concurrencia a partir del código fuente de un programa. A continuación, se enumeran las tareas que se deben materializar:

- Modificación de las señales de tráfico con el objetivo de mostrar su contenido por ambos lados. Se pretende que la información que aportan se visualice desde cualquier perspectiva.
- Modificación de la metáfora de una caja como representación para almacenar el valor de las variables. Se pretende añadir como alternativa un túnel con un tramo de carretera.
- Representación del concepto de proceso o hilo de ejecución.
- Representación del concepto de punto de ruptura (o *breakpoint* en inglés).

2.2.2 Diseño y codificación de un sistema de visualización dinámica

Uno de los objetivos más importantes de este trabajo es introducir un nuevo modo de visualización que permita comprender mejor la ejecución de un programa. Para ello, este subobjetivo pretende crear un entorno de visualización dinámica desde dos puntos de vista: i) generación dinámica del entorno virtual a partir del código fuente y ii) ejecución dinámica del propio programa. Este enfoque será cubierto mediante el uso de un vehículo, cuya función será la de moverse sobre una representación la cual mantiene una correspondencia entre carreteras y señales de tráfico con sentencias de código y estructuras de control. A continuación se describen las tareas necesarias para cumplir con este subobjetivo:

- Despliegue del entorno de depuración de Eclipse programáticamente para evaluar sentencias de forma dinámica.
- Implementación de las funcionalidades principales de un depurador como *step over* y *step up to a point*.
- Diseño e implementación de una estructura que permita la creación dinámica de grafos cíclicos dirigidos con la que guiar al vehículo sobre la representación gráfica.
- Generación dinámica del entorno virtual por el que circula el vehículo y que mantiene una correspondencia con la capa lógica del sistema (depurador).
- Uso adecuado de patrones de diseño y otras buenas prácticas de programación que proporcionen mecanismos para ayudar a construir una arquitectura escalable y mantenible.

2.2.3 Propuesta para la mejora en el proceso de aprendizaje de la programación concurrente

El aprendizaje de la programación concurrente y en tiempo real supone un esfuerzo importante para los estudiantes de programación acostumbrados a desarrollar programas con un único proceso, es decir, un único programa en ejecución. Este problema surge al utilizar diferentes procesos e hilos que necesitan de su comunicación para sincronizarse correctamente. Por lo tanto, el objetivo que se pretende conseguir es que este sistema soporte la ejecución de varios procesos e hilos en un mismo programa actuando simultáneamente, donde la RA toma un papel importante al permitir hacer un seguimiento personalizado de cualquiera de las entidades activas en ejecución.

Como primera aproximación para tratar de resolver esta problemática, se propone la ampliación de la metáfora existente para representar términos específicos del paradigma citado con elementos de circulación vial. En este enfoque todos los elementos de la metáfora mantienen su función a excepción del vehículo, el cual pasa a mantener una correspondencia con los procesos o hilos de un programa. Cada vehículo se moverá por la representación, simulando actuar como una entidad en ejecución. De esta forma, se espera facilitar la comprensión de conceptos y mecanismos de sincronización al reducir el nivel de complejidad de esta tarea. A continuación, se enumeran las tareas que se deben realizar:

- Representación del concepto de sección crítica[1].
- Representación del concepto de semáforo[2].
- Representación del concepto de operación *signal*[3].

2.2.4 Evaluación y caso de estudio de la propuesta

Para evaluar la propuesta anterior se pretende llevar a cabo una experiencia con estudiantes de la asignatura de Sistemas Operativos II (SSOOII) del tercer curso del Grado en Ingeniería Informática. Esta prueba se encuentra dividida en dos partes, una para conocer el nivel de los estudiantes y otra para comprobar si la metáfora ayuda considerablemente a aquellos que más les cuesta, aunque también pretende servir como refuerzo a estudiantes que no presentan en principio dificultades. En esta segunda parte, se incluye un caso de estudio donde los estudiantes tienen que resolver una actividad y responder preguntas asociadas a esta actividad. A continuación, se enumeran las tareas que se deben efectuar:

- Diseño de la evaluación para los estudiantes de SSOOII.
- Realización de un pre-test con preguntas sobre sus conocimientos en programación.
- Realización de un post-test con preguntas para medir las dimensiones de comprensión y adecuación de cada una de las representaciones propuestas.

[1] Fragmento de código de un programa a la que solo un proceso o hilo puede acceder.
[2] Abstracción con el cual se restringe o se permite el acceso a un recurso compartido.
[3] Operación que incrementa el valor de una semáforo.

- Caso de estudio para valorar, objetivamente, si la propuesta es útil para comprender programas con concurrencia.

Capítulo 3

Antecedentes

En este capítulo se recoge un estudio sobre el que se asienta este trabajo y que constituye su base de conocimiento. Previamente, este trabajo parte de la realización de una extensa investigación acerca del estado actual de los sistemas para la visualización de programas y algoritmos, entornos de desarrollo extensibles, dispositivos de RA actuales, algoritmos de seguimiento y la utilización de metáforas en entornos para el aprendizaje de la programación. Todo este análisis se recoge en [GP18]. Sin embargo, tras la ampliación de este trabajo, surgen nuevas líneas de investigación que son de interés estudiar. En primer lugar, se estudia la visualización dinámica de programas, haciendo una comparativa de los sistemas más populares durante la última década. A continuación, un recorrido por la literatura para conocer la efectividad de la visualización de programas. Después, una introducción a la RA, haciendo especial énfasis en las diferentes técnicas de integración de contenido virtual: con marcas o sin marcas. Además, se describe de forma sintetizada el trabajo anteriormente citado, explicando aquellos aspectos más importantes de su arquitectura y destacando la característica de la visualización estática de programas. Por último, este capítulo concluye con los beneficios que aporta el trabajo expuesto en este proyecto, realizando una comparativa con los temas aquí tratados.

3.1 Visualización dinámica de programas

La visualización dinámica de programas es un término que se engloba dentro de la visualización de programas según la taxonomía definida por Myers [Mye90], el cual se refiere a sistemas que muestran gráficamente información de un programa en ejecución. Estas visualizaciones ilustran desde código o estructuras de datos hasta algoritmos de un programa. En esta, sección se presenta la evolución y el estado actual de esta línea de investigación, donde se revisan algunos de los sistemas más representativos ubicados en la literatura. Además, se menciona brevemente las características más notables de estos sistemas y se hace especial hincapié en los beneficios que éstos aportan en el proceso de aprendizaje con respecto a los métodos de aprendizaje tradicionales.

Un ejemplo representativo que trata de representar las estructuras de datos de un programa (p. ej., tipo de una variable, un registro, etc.) es el trabajo de Rowe y Thorburn [RT00], en el

cual se describe una herramienta que permite trazar la ejecución de un programa en C como si de un depurador se tratase, pero aún más detallado. Dicha herramienta permite seguir la traza del valor de cada variable durante la ejecución y proporcionar información sobre el estado actual de la memoria.

VINCE, nombre atribuido a la herramienta, consiste de un panel bidimensional con diferentes ventanas (ver Figura 3.1). Arriba a la izquierda se muestra una pequeña ventana donde el usuario puede escribir o pegar un código en el lenguaje C. A la derecha, un cuadro con el que elegir ejercicios diseñado para mostrar aspectos de programación. En la mitad inferior se presenta una rejilla, donde cada cuadrado representa un *byte* en memoria.

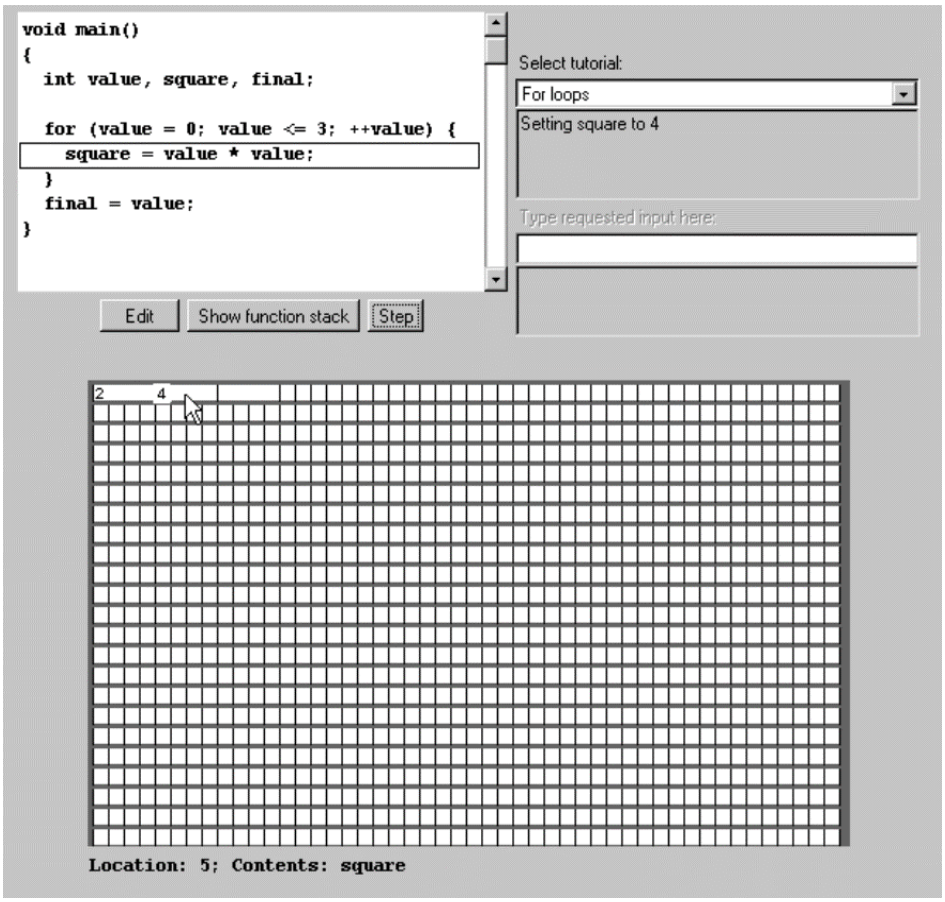


Figura 3.1: Interfaz de la herramienta VINCE [RT00]

Además de las características técnicas, este trabajo presenta una evaluación con estudiantes de primer año que asisten a un curso introductorio a la programación. En esta evaluación, los autores tratan de demostrar el beneficio que supone utilizar VINCE a la hora de comprender el funcionamiento de un programa codificado en lenguaje C. En este prueba, los estudiantes fueron divididos en dos grupos espejo, en el sentido que para cada estudiante con una habilidad dada en un grupo debía haber un estudiante en el otro con habilidades similares.

Tras la finalización de la prueba, los resultados arrojados mostraron que el uso de VINCE

presenta un cierto beneficio en el proceso de aprendizaje. Concretamente, en términos de la evaluación, aquellos estudiantes que utilizaron la herramienta mejoraron la comprensión del funcionamiento de programas frente aquellos que utilizaron los medios de aprendizaje clásicos.

Respecto a trabajos que tratan de mostrar gráficamente partes del código que están siendo ejecutadas es relevante mencionar el de Ben-Ari *et al* [BABL+11] con su herramienta Jeliot 3[1]. En este trabajo se hace una revisión de las diferentes versiones por las que ha pasado esta herramienta, destacando las mejoras que se han incorporado a lo largo del tiempo. Este sistema se centra en el estudio de conceptos como son la asignación de variables, E/S y flujo de control, entre otros. Además, permite a los estudiantes ver el código fuente de un programa y mostrar entidades animadas durante la ejecución de un programa.

Jeliot 3 está pensado totalmente para estudiantes con muy pocas nociones de programación. La interfaz de la herramienta es lo más sencilla posible, constituida por una ventana dividida en dos paneles (ver Figura 3.2). El panel de la izquierda constituye el área del programa a animar, mientras que el panel derecho muestra las entidades del algoritmo animado. Además, existe una zona en la parte inferior de la herramienta, tanto para mostrar la salida del programa como para ajustar la velocidad de la animación.

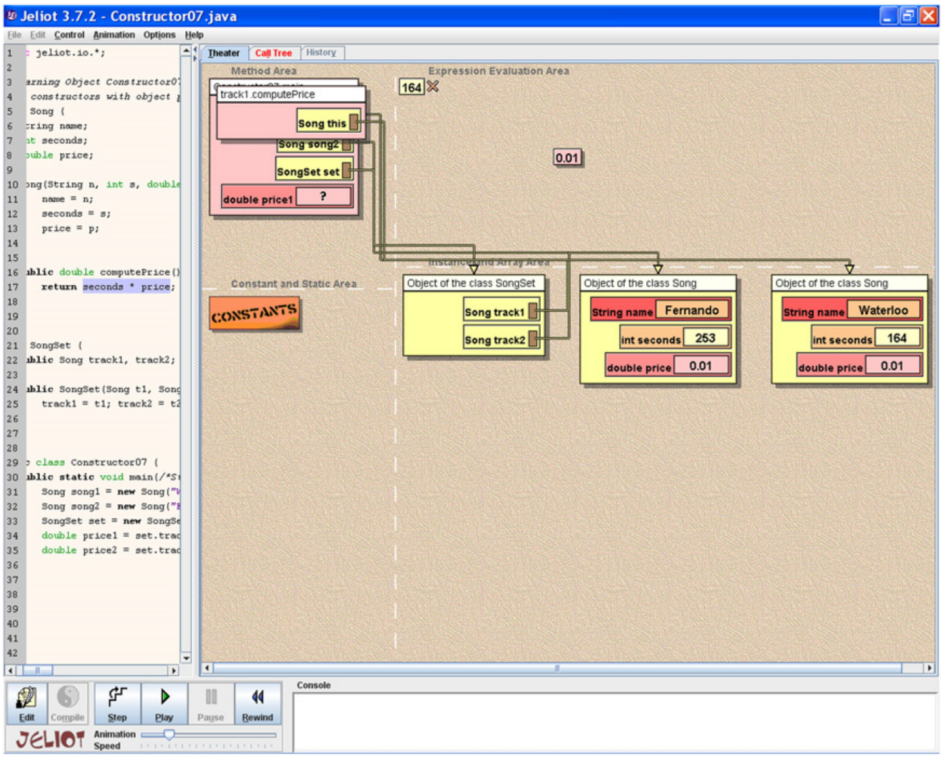


Figura 3.2: Visualización dinámica de un fragmento de código en Jeliot 3 [BABL+11]

Aparte, este trabajo resume en una variedad de contextos la extensa investigación pedagógica realizada sobre el uso de Jeliot en el aula. Entre los resultado obtenidos de las eva-

[1] http://cs.joensuu.fi/jeliot/downloads/jeliot372.php

luaciones destacan los siguientes: adquisición y uso de un mejor vocabulario de términos de programación; incremento en la atención de los estudiantes en clase; o el aumento del nivel de compromiso entre estudiantes y la herramienta, entre otros.

Similarmente, Pérez en [PC08] aborda el problema de la recursividad mediante un sistema 2D que incorpora el enfoque de visualización dinámica de algoritmos.

SRec[2], herramienta fruto del trabajo citado, soporta un extenso conjunto de representaciones, tales como trazas, pila de llamadas, árboles recursivos y grafos de dependencia. Cada representación es usada en función del problema a resolver, creando una animación automática a medida que la ejecución del algoritmo avanza. Además, SRec proporciona interacción con las visualizaciones, permitiendo filtrar la cantidad de datos a visualizar, cambiar el orden relativo de los datos y buscar datos específicos en la visualización, entre otras funcionalidades. Todo esto se realiza desde la interfaz de la herramienta (ver Figura 3.3), la cual se divide en varias zonas para representar el código fuente del algoritmo a animar, la traza del algoritmo, ajustes de la animación y la visualización del algoritmo que se está ejecutando, entre otras funciones.

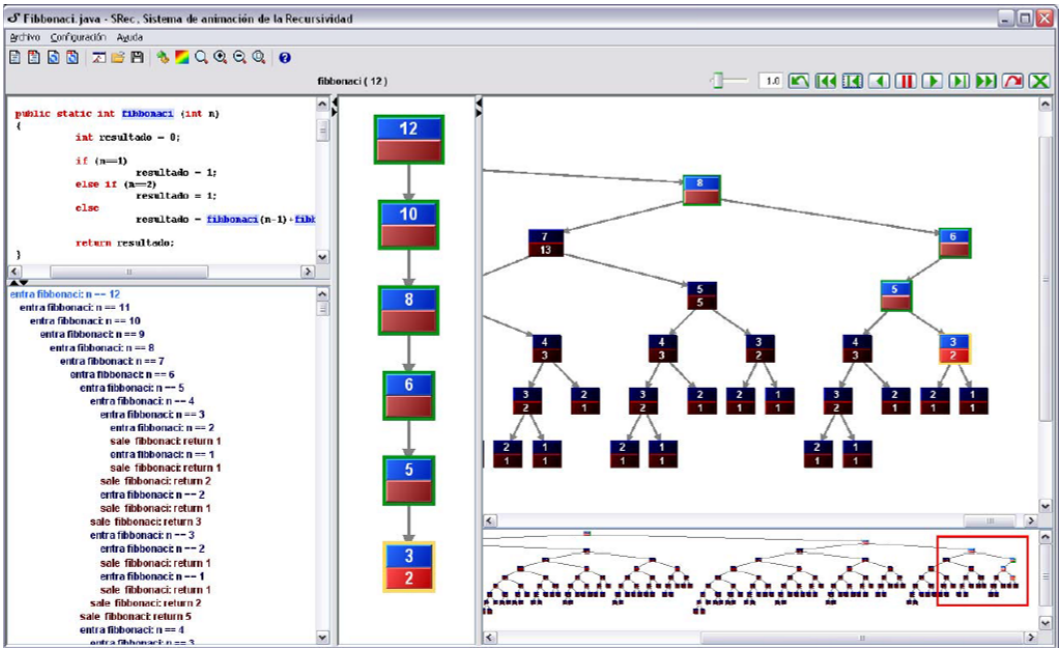


Figura 3.3: Visualización del algoritmo de *fibonacci* con SRec [PC08]

De este trabajo surgen varios estudios donde se mencionan las ventajas de esta herramienta. En [PC11] se realizaron 5 evaluaciones para medir tanto la usabilidad como su ayuda en el ámbito docente. Dicho trabajo arroja resultados que demuestran que SRec es de utilidad en este último, siendo acompañado por un profesor en la explicación de conceptos de recursividad. Sin embargo, estudiantes critican su complejidad a la hora de utilizarse en ciertos escenarios.

[2] `http://www.lite.etsii.urjc.es/tools/srec/srec-download/`

Por otro lado, el uso de visualizaciones también es usado en el contexto del aprendizaje de la Programación Orientado a Objetos (POO). Yang *et al* en [YLHC15] muestran una plataforma web educativa para la enseñanza de este paradigma de programación. A través de un navegador web, los estudiantes pueden desarrollar sus programas en lenguaje Java sin necesidad de instalar *software* o *plug-ins* de terceros. De esta forma, ellos son capaces de programar en cualquier lugar y en cualquier momento, siendo solo necesario contar con un dispositivo con acceso a internet para acceder a la plataforma educativa.

Esta herramienta se apoya en una arquitectura de red cliente-servidor. El cliente envía el código fuente que se desea visualizar a través de un navegador web. Mientras, el servidor es quien se encarga de extraer esa información (variables, parámetros, tipos, etc.), la cual se convierte a *bytecode* para generar una visualización en tiempo de ejecución. En la Figura 3.4 se presenta este flujo de información de un modo más detallado.

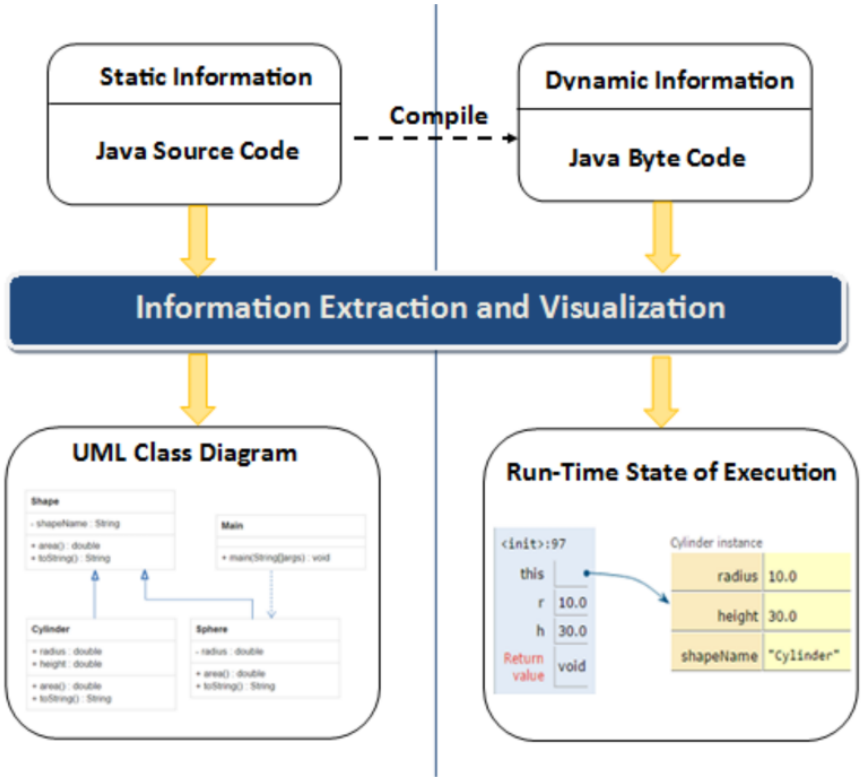


Figura 3.4: Visión general de la plataforma [YLHC15]

Finalmente, los autores concluyen que con esta herramienta esperan reducir la carga de trabajo cognitivo de los estudiantes programadores, aparte de mejorar la comprensión de los conceptos de programación y diseño orientados a objetos.

En el campo de los gráficos 3D, existen también una variedad de trabajos que usan este tipo de técnicas para facilitar el aprendizaje de la programación.

Un ejemplo de este caso es el trabajo de Milne y Rowe [MR04] con OGRE, una herramienta para la visualización de programas desarrollada específicamente para abordar el

problema de la gestión de memoria en el lenguaje C. En este trabajo, a través de gráficos 3D, se estudia qué ocurre cuando un objeto se almacena en memoria, lo que sucede en el caso de que los objetos se utilicen en sentencias de asignación o cuando son pasados de un método a otro.

A diferencia de la herramienta VINCE, OGRE adopta una visión de alto nivel, dividiendo un programa en ejecución en los diferentes ámbitos (o *scope* en inglés) que se compone (p. ej., un método). De este modo, OGRE trata de evitar saturar al usuario con excesiva información.

Los recursos gráficos que utiliza OGRE son: planos, que representan el *scope* de un programa en ejecución; cubos, esferas y conos para representar objetos y variables; flechas para representar la conexión entre objetos; y cilindros que actúan como tuberías para mostrar el flujo de datos entre objetos. En la Figura 3.5 se muestra un ejemplo de las capacidades de esta herramienta.

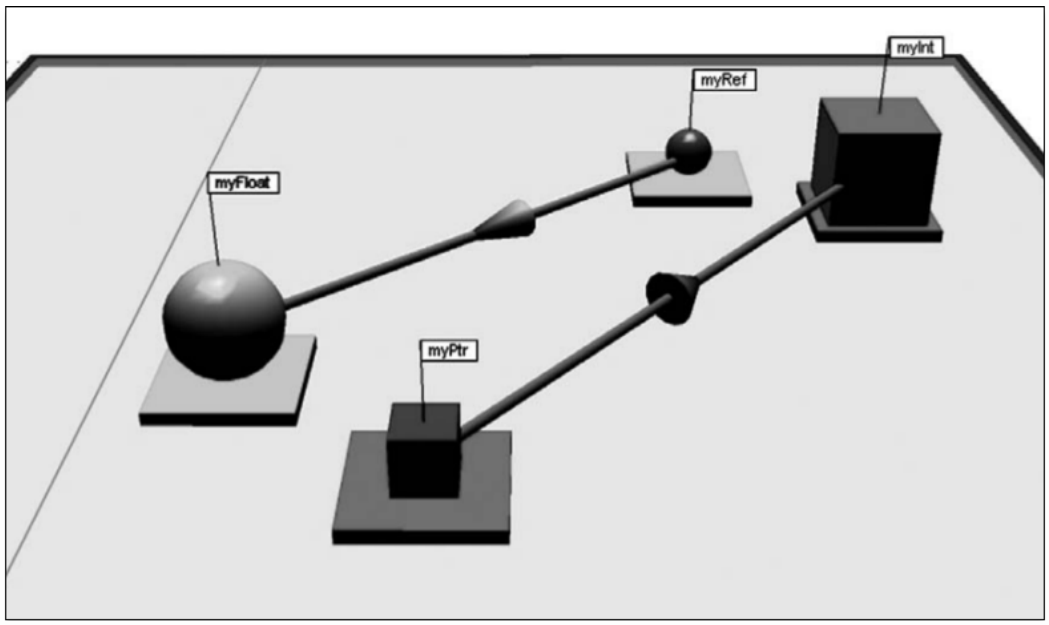


Figura 3.5: Ejemplo de referencia de variables en OGRE [MR04]

Este trabajo termina con una evaluación a estudiantes de programación, cuyo objetivo es demostrar que la herramienta ayuda a entender cómo se gestiona la memoria durante la ejecución de un programa. Para ello se realizaron dos grupos, uno que sirvió como grupo de control, y otro que se utilizó como grupo de prueba. El estudio concluye que OGRE presentó beneficios en los estudiantes respecto a temas relacionados con la gestión de memoria. Aparte de esto, estudiantes y profesores aportaron opiniones respecto a la herramienta, quienes manifestaron que los gráficos 3D aportan beneficios en la comprensión de programas, siendo un factor clave la libertad de movimiento por la escena, así como la posibilidad de tener diferentes perspectivas que permiten obtener una mejor visualización.

Jimenez-Diaz *et al* presentan en [JDGCGA12] Role-Play, un entorno virtual de juego de rol para la enseñanza de la POO. Este entorno destaca por su originalidad, ofreciendo una novedosa metáfora visual que usa avatares y animaciones 3D para representar conceptos de programación, tales como objetos, clases y paso de mensajes. Esta información, que se recoge del programa que se quiere explorar, es mostrada en un escenario, el cual puede ser jugado de dos formas diferentes: individual o multijugador. El escenario de un jugador solo permite familiarizar a éste con conceptos básicos, mientras que el multijugador permite la discusión y colaboración de los usuarios sobre la interacción entre los objetos del escenario. A modo de ejemplo, la Figura 3.6 muestra un escenario en el que participan dos jugadores.

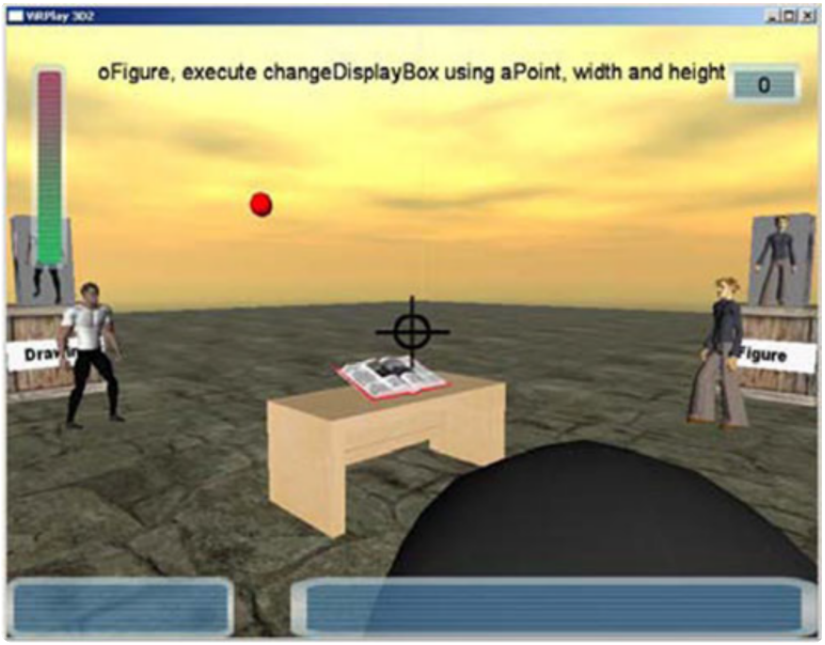


Figura 3.6: Entorno multijugador en RolePlay donde dos jugadores trabajan colaborativamente para resolver un escenario [JDGCGA12]

Este entorno ha sido evaluado en dos ocasiones. Por un lado, se realizó una prueba con estudiantes para medir el grado de usabilidad y adecuación de la metáfora. Por el contrario, se estudió el impacto que implica su uso por parte de los estudiantes en el aula. Respecto al primer caso, la aceptación en general fue buena, considerando los participantes que el entorno es entretenido y fácil de usar. Además, aportaron comentarios a la metáfora propuesta, destacando que es adecuada para el aprendizaje de la POO. En cuanto al segundo, los resultados obtenidos no demostraron una mejora en el aprendizaje, pero sí se percibió una motivación en los estudiantes durante su uso.

Similarmente, otro enfoque para representar algoritmos usando técnicas de visualización es la presentada en [MOM18], cuyo trabajo se constituye como un entorno virtual inmersivo para la enseñanza de conceptos de programación concurrente mediante una metáfora de actores y cajas 3D.

Este entorno, denominado IDEA, se presenta como un depurador en Realidad Virtual (RV)

cuya característica principal es la de visualizar una secuencia parcialmente ordenada de mensajes asíncronos intercambiados durante la ejecución de un programa. IDEA además proporciona a los usuarios una interfaz 3D para establecer puntos de ruptura (o *breakpoints* en inglés), navegar sobre la traza (hacia adelante o hacia atrás) o consultar el historial de mensajes durante la ejecución, entre otras características.

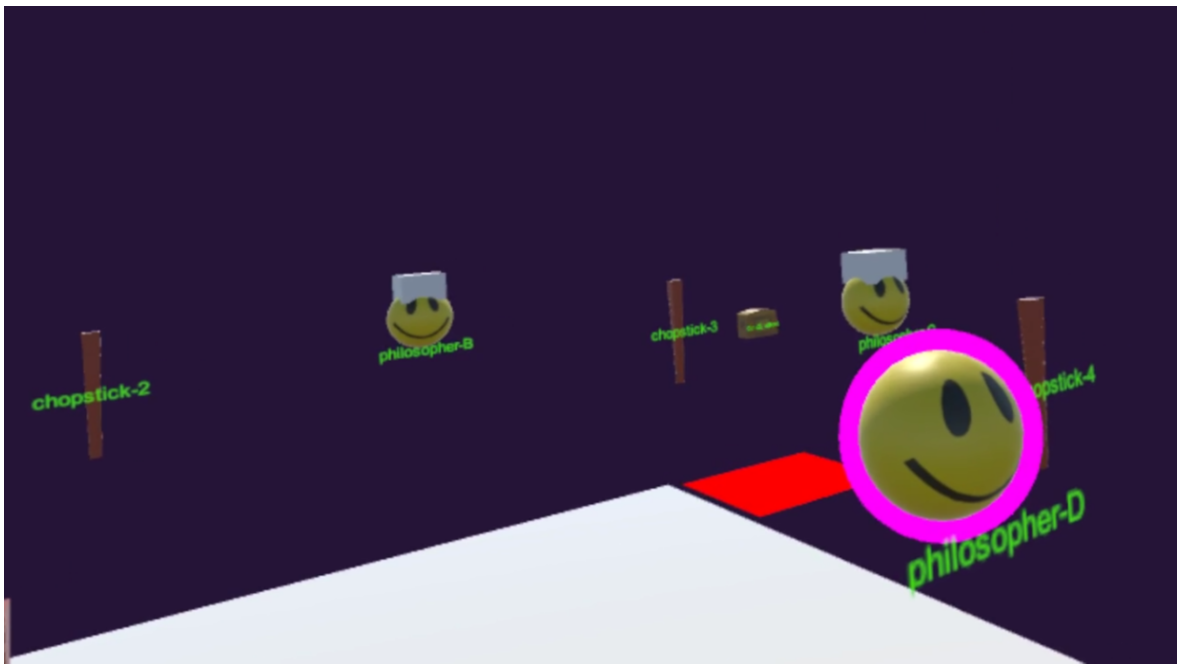


Figura 3.7: Ejemplo del problema de los filósofos comensales usando IDEA [MOM18]. El actor colorado indica que éste está involucrado en la acción actual

3.2 Efectividad de la visualización de programas

El proverbio «una imagen vale más que mil palabras» muestra por sí sola la ventaja que supone representar gráficamente conceptos abstractos difíciles de explicar [LS87]. Nuestro sistema visual está optimizado para procesar información en más de una dimensión. Por lo tanto, la creación de visualizaciones para mostrar la información de la estructura de un programa o de su ejecución parece adecuado para aquellos usuarios poco experimentados en la tarea de programar [Mye90].

El uso de animaciones puede resultar que algoritmos complejos se conviertan en otros más comprensibles y menos intimidatorios. En [KST01] se realizó un estudio con estudiantes para afirmar esta hipótesis, cuyo resultado demuestra que la presencia de animaciones involucra a los estudiantes en el proceso de aprendizaje, así como crear una experiencia de aprendizaje en lugar de un reto. Aparte, esta forma de visualización supone un impacto notable en las dos dimensiones de la motivación de los estudiantes: intrínseca (acciones llevadas a cabo por propia voluntad o deseo) y extrínseca (acciones que implican una recompensa externa) [VIHLPV17].

Hundhausen *et al* en [HDS02] mantienen un enfoque más controvertido. Según los autores, el logro educativo se consigue mediante una correcta utilización de las visualizaciones en la actividad de aprendizaje, no por su calidad. Además, Naps *et al* exponen en [NRA+02] la dificultad para el profesor de crear y diseñar visualizaciones que mantengan a los estudiantes interesados en ellas durante las clases.

No obstante, la irrupción de tecnología inmersiva y el aumento de dispositivos que mejoran la visualización de representaciones gráficas pretenden romper con este hecho [BBF+14]. Concretamente, la RA es una tecnología que por las ventajas que ofrece es dada a usarse en el ámbito académico. Por esta razón, un caso de estudio ha sido identificar los beneficios que esta tecnología aporta al proceso de aprendizaje. Diegmann en [DSKEB15] realiza una revisión por la literatura sobre este asunto, en cuyo trabajo destaca que entornos educativos con RA dan lugar a ciertos beneficios, como la motivación, atención, concentración y satisfacción. Además, resulta de gran interés mencionar los mecanismo naturales de interacción que soporta esta tecnología, los cuales permiten generar experiencias enriquecedoras para el usuario por la facilidad de manipular contenido 3D [DD14].

3.3 Realidad Aumentada

La RA es un campo de estudio que se encarga de analizar las técnicas que permiten integrar en tiempo real contenido digital con el mundo real. Según la taxonomía descrita por Milgram y Kishino [MTUK95], los entornos de Realidad Mixta (RM) son aquellos en los que «se presentan una mezcla de clases de objetos (tanto reales como virtuales) de forma conjunta en una única pantalla». Esto abre un abanico de posibilidades, dando lugar a diferentes tipos aplicaciones, tal y como se muestra en la Figura 3.8.

Más tarde, en 1997, Azuma publicó un estudio [Azu97] en el define esta tecnología donde el usuario puede ver el mundo real compuesto de objetos virtuales. Adicionalmente, este trabajo define las características que un sistema de RA debe incorporar, las cuales son:

1. Combina mundo real y virtual.
2. Interactivo y en tiempo real.
3. Alineación 3D.

A diferencia de la RV, donde el usuario interactúa en un mundo totalmente virtual, la RA se ocupa de generar capas de información virtual que deben ser correctamente alineadas con el mundo real para lograr una sensación de integración correcta.

Sin embargo, esta integración no es trivial, dado que es necesario calcular la posición relativa de la cámara con respecto de la cámara virtual para generar imágenes virtuales perfectamente alineadas con la imagen real [GVAC13]. Esto es lo que comúnmente se conoce como el problema del registro.

Figura 3.8: Taxonomía del concepto de RM de Milgran y Kishino

No obstante, este problema puede resolverse mediante técnicas de seguimiento visual (o *visual tracking* en inglés). En [Mar07] se presenta una taxonomía general de este tipo de técnicas. Por un lado se encuentra *Bottom-Up*, que trata de obtener los seis grados de libertad de la cámara (tres de posición y tres de rotación) mediante la geometría del objeto. Por el contrario existe el método *Top-Down*, el cual estima la posición de la cámara dependiendo del contexto de la escena. En esta sección, se trata la primera técnica al basarse este sistema en una de sus clases.

La primera clase a destacar que se engloba en esta técnica es el **seguimiento basado en marcas** (o *marker-based tracking* en inglés), cuya idea es la de obtener los seis grados de libertad mediante marcas cuadradas o circulares, normalmente de papel o cartón, con diferentes patrones dibujados en blanco y negro sobre una cara. En la Figura 3.9 se muestran varios ejemplos de este tipo de marcas.

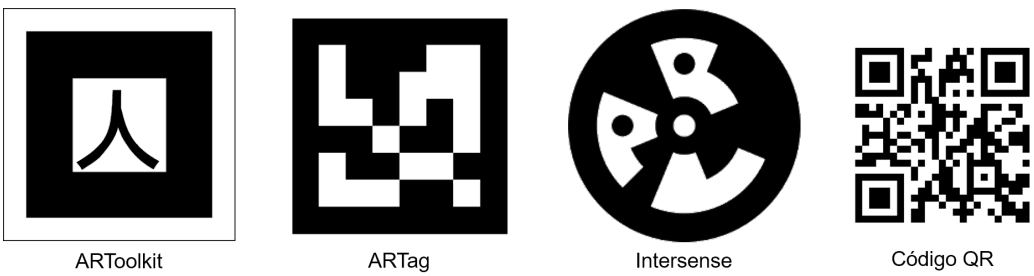


Figura 3.9: Ejemplos de marcas cuadradas y circulares

ARToolkit[3] es uno de los sistemas basado en marcas más extenso y usado para la creación de contenido aumentado introducido por Kato y Billinghurst en [KB99]. Este método analiza fotograma a fotograma, realizando técnicas de binarizado y normalización para detectar la existencia de marcas en el entorno y su posición. ARTag es un sistema mejorado del anterior identificando y verificando marcas debido a la robustez ante cambios de iluminación y oclusión parcial [Fia05]. Intersense es un tipo diferente de marca, circular, que suele ser robusto cuando aumenta el número de marcas empleadas [NF02]. Por último, los códigos QR también se pueden clasificar como un tipo de sistema basado en marcas, ya que gracias a su nube de puntos bidimensionales permiten ser identificados fácilmente en un entorno complejo.

Por otro lado, se encuentra la clase **sin marcas** (o *markless tracking* en inglés) que se apoya en técnicas de visión por computador para detectar características naturales del espacio físico, tales como las esquinas de una mesa, el contorno de las líneas de una caja, etc. A continuación se describe brevemente las diferentes técnicas que se engloban en esta clase.

- **Estructuras planas**. Esta técnica estima la posición relativa de la cámara usando áreas planas fácilmente detectables en la escena, tales como suelo, pared, techo, etc. Sin embargo, esto requiere de potentes recursos computacionales comparado con enfoques basados en marcas. En la literatura se encuentran algunos trabajos relevantes como son [SB02], en el cual se presenta un método que utiliza dos o más planos de la escena real para el seguimiento sin marcas y [JD02], que estima la posición de un plano de la escena mediante aproximación por hiperplano. Este método es relativamente común en el ámbito de la visión por computador.
- **Basado en modelos**. Este método se basa en utilizar modelos asistidos por ordenador (o Computer-Aided Design (CAD) en inglés) de partes del entorno real, normalmente un objeto, para detectar aristas o puntos claves en la imagen. La posición de la cámara es relativa al centro del objeto. En este método se consideran dos problemas: cuando las aristas del objeto son confundidas con el entorno o diferentes aristas fallan debido a la forma del objeto. En la literatura, el trabajo de Drummond y Cipolla [DC02] solventa la primera situación asignando un peso a cada arista detectada en el espacio de búsqueda, mientras que en el trabajo de Vachetti *et al* [VLF04] aborda ambos problemas, añadiendo nuevos puntos de observación durante el seguimiento para minimizar el error ante grandes cambios.
- **Escenas sin restricciones**. Este método trata el problema de aquellos escenarios en los cuales no es posible conocer previamente estructuras planas o modelos, como por ejemplo, un entorno al aire libre complejo que no puede ser modificado con antelación. Varios trabajos han explotado lo que se conoce como limitación epipolar[4] (o

[3] http://www.hitl.washington.edu/artoolkit/

[4] https://en.wikipedia.org/wiki/Epipolar_geometry

epipolar constraint en inglés) para tratar con escenarios en los que no se pueden imponer restricciones geométricas. Algunos trabajos que abordan estos problemas son el de Chia *et al* [CCP02] donde proponen un sistema que realiza el seguimiento de puntos clave y el cálculo de la limitación epipolar entre el fotograma actual y el fotograma clave. Nistér *et al* [NNB04] presentan un marco de trabajo que combina la limitación epipolar entre fotogramas continuos y discontinuos con métodos basados en mapeo de entornos 3D. No obstante, la utilización de estas técnicas suponen un alto coste computacional, el cual está siendo reducido mediante la técnica de seguimiento por detección (o *tracking-by-detection* en inglés) utilizada en trabajos como [SL04] y [LLF05].

3.4 AsgAR y visualización estática

AsgAR [GP18] es una plataforma compuesta de dos sistemas para la visualización estática de programas y algoritmos. Esta plataforma se apoya en una arquitectura de red cliente-servidor, donde el servidor es un plug-in implementado en Eclipse para automatizar la elaboración de representaciones gráficas. El cliente, un sistema empotrado en el dispositivo de RA de Microsoft capaz de representar gráficamente programas alineados con el espacio físico real.

Este tipo de representaciones se han llevado a cabo mediante la elaboración de una serie de modelos basados en la notación ANGELA. Cada uno de ellos se asocia con una instrucción de programación, tratando de minimizar la abstracción que requiere la programación. En la Figura 3.10 se muestra el conjunto de metáforas asociadas a las instrucciones de programación descritas a continuación:

- **Definición de función**. Representado en la figura por una señal de tráfico y un fragmento de carretera vertical (a). En esta representación, la metáfora proporciona información sobre el nombre de la función y sus argumentos. Cada visualización comienza con esta representación gráfica.
- **Instrucción bucle**. Representado en la figura por una señal de tráfico y una rotonda (b). A diferencia de la definición de función, la señal de tráfico es usada para mostrar la condición que decide si el cuerpo del bucle se ejecuta o no. Mientras, la rotonda representa el concepto clave de la metáfora, donde un vehículo puede viajar indefinidamente hasta cumplir con el número de acciones a repetir. La salida este de la representación, vista desde una perspectiva frontal, indica la ejecución del cuerpo del bucle. Por el contrario, la salida sur representa el fin de ejecución del fragmento de código incluido en el mismo.
- **Instrucción condicional**. Representado en la figura por una señal de tráfico y una bifurcación (c). Al igual que con la instrucción bucle, la señal de tráfico muestra la condición que decide qué rama de la instrucción se ejecuta. La rama de la izquierda,

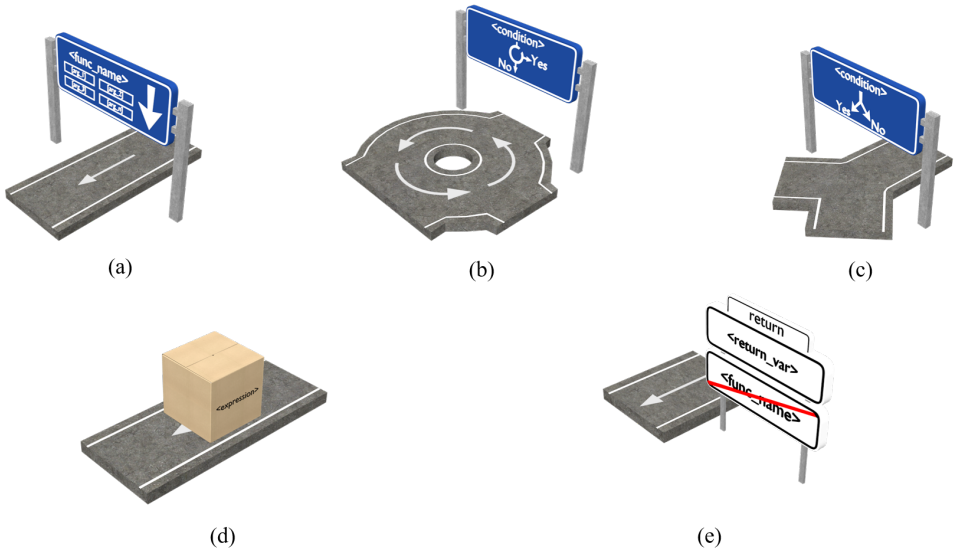


Figura 3.10: Conjunto de representaciones gráficas que definen la notación ANGELA: (a) definición de función, (b) instrucción bucle, (c) instrucción condicional, (d) evaluación de expresión y (e) función de retorno.

vista desde una perspectiva frontal, refleja la ejecución del fragmento de código tras una evaluación a verdadero de la condición. Por el contrario, la rama derecha ejecuta el cuerpo tras evaluar la condición a falso.

- **Evaluación de expresión**. Representado en la figura por una caja que descansa sobre un fragmento de carretera vertical (d). En esta representación, la metáfora proporciona información sobre llamadas a función, declaraciones, asignaciones, inicializaciones, etc.

- **Función de retorno**. Representado en la figura por una señal de tráfico y un fragmento de carretera vertical (e). En esta representación, la metáfora proporciona información sobre el valor que es devuelto (en el caso que la función devuelva un valor) y el nombre de la función a la cual pertenece la instrucción.

Las representaciones, excepto la función de retorno, incluyen mecanismos de interacción avanzada, desplegando un panel sobre la señal de tráfico o caja para mostrar información extra sobre la instrucción.

Todo lo anteriormente discutido es en relación al cliente, el cual se despliega en el dispositivo de RA. Por el contrario, respecto al servidor, se diseñó una interfaz en Eclipse con un conjunto de funcionalidades para facilitar la interacción entre el usuario y el sistema. Éstas permiten al usuario conectarse con el dispositivo de RA a través de un código QR, seleccionar un método para su posterior visualización, observar la conversión del código fuente

de Java a JavaScript Object Notation (JSON), así como ver y retransmitir el contenido del dispositivo.

Estas funcionalidades son descritas en detalle a continuación, las cuales aparecen marcadas en la interfaz que se muestra en la Figura 3.11.

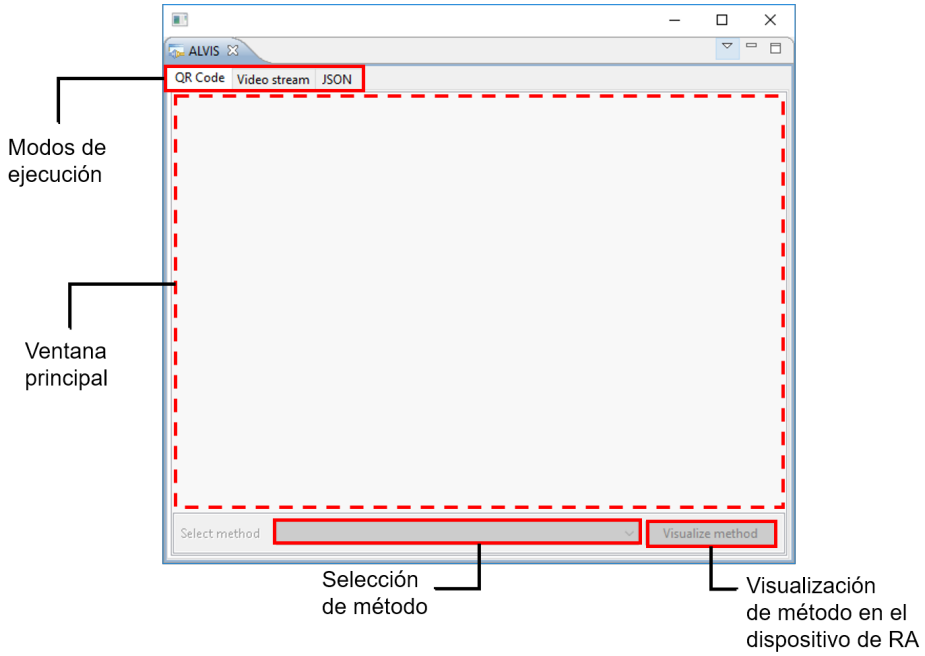


Figura 3.11: Diseño de la interfaz gráfica del plug-in de visualización

- **Código QR**. Representado en la ventana como *QR Code*. Muestra en la venta principal la matriz de puntos bidimensional que almacena la dirección de red del servidor. Con este código el usuario se conecta al *plug-in* para poder convertir código Java a una metáfora de carreteras y señales de tráfico.
- **Retransmisión**. Representado en la ventana como *Video Stream*. Permite ver en tiempo real el contenido generado por el dispositivo de RA.
- **JSON**. Muestra la conversión del fragmento de código a visualizar en un formato (JSON) fácilmente interpretable por el dispositivo de visualización.
- **Selección de método**. Cuadro de lista encargado de recoger el nombre de los métodos de la clase activa en el editor de Eclipse.
- **Visualización de método**. Representado en la ventana como *Visualize method*. Este botón es el que el usuario debe clicar para generar una visualización 3D a partir del método seleccionado en el cuadro de lista anterior. Un archivo en formato JSON es generado y enviado al dispositivo de RA para crear la representación gráfica.

Este entorno ha sido integrado en el *plug-in* COLLECE-2.0[5], una herramienta que provee

[5]http://blog.uclm.es/grupochico/proyecto-iapro/collece-2-0/

de un conjunto de características para facilitar la programación mediante la participación, colaboración remota y síncrona. Este nuevo plug-in aporta a COLLECE 2.0 nuevas funcionalidades, como son el análisis del código fuente de un programa, gestión de la conexión de red con el dispositivo de RA y la retransmisión en tiempo real del contenido 3D a los usuarios conectados a una sesión de COLLECE.

Para el análisis de código fuente de un programa, este plug-in utiliza el *framework* de Eclipse Abstract Syntax Tree (AST), el cual permite analizar cada una de las instrucciones del lenguaje relacionados con los elementos de la metáfora propuesta (ver Figura 3.10). Este análisis trata de extraer el código fuente de la clase activa en el editor (1), analizar el mismo a través del *framework* anterior (2), procesar el árbol generado (3) y convertir las instrucciones Java en el formato de datos especificado (4). Todo esto se muestra a modo de resumen en la Figura 3.12.

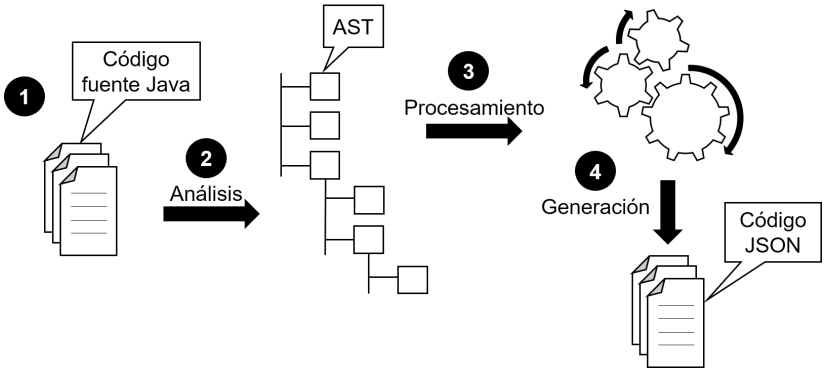


Figura 3.12: Síntesis del procesamiento del código fuente de Java

En este formato de intercambio de datos (JSON) se almacena la información de cada instrucción del método a visualizar, con el objetivo de facilitar su representación en el dispositivo de visualización. Así, éste reduce su carga computacional, enfocando su trabajo en el renderizado de las carreteras. En la Figura 3.13 se puede apreciar la relación que tiene cada representación gráfica con la conversión de una instrucción Java en el formato JSON.

Una vez completado el procesamiento y generado el archivo JSON correspondiente al método a visualizar, éste es enviado al dispositivo de RA a través de *sockets* de red Transmission Control Protocol (TCP). Al comienzo del archivo, una sentencia define el propósito del mensaje, el cual puede ser: JSON o *streaming*. Dependiendo del tipo de mensaje, el dispositivo realiza una u otra acción.

En cuanto al establecimiento de la conexión de red, el usuario escanea el código QR que aparece en la vista mediante el dispositivo de visualización. De esta manera, cliente y servidor mantienen una comunicación para establecer la visualización estática de programas o algoritmos.

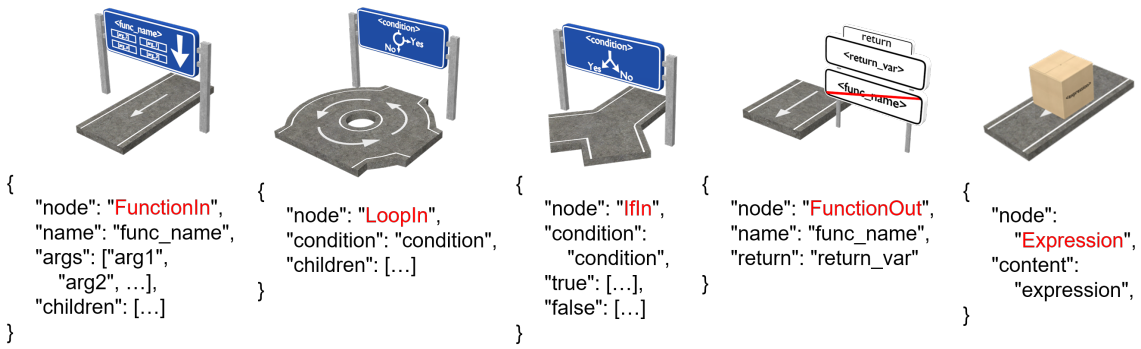


Figura 3.13: Formato JSON relacionado con cada metáfora del conjunto de representaciones gráficas

3.5 Conclusiones

La característica diferenciadora del enfoque presentado en este trabajo es que, con la ayuda de un dispositivo de RA, se visualiza la traza y el código fuente de un programa mediante el uso de un conjunto de representaciones gráficas, que asocian términos de circulación vial con conceptos específicos de programación.

Su explotación en un entorno de RA permite que usuarios puedan ver el código fuente de un programa desplegado de forma visual dirigiendo la mirada por la representación, quien tiene total libertad de movimiento al utilizar unas gafas que mantienen una correspondencia entre su posición y la de la visualización. Además, esta técnica permite mejorar el realismo que conlleva la propia analogía con los elementos físicos que representa.

Incluso, este planteamiento da lugar a visualizar programas donde exista concurrencia, pues es posible mantener un seguimiento individualizado de los procesos o hilos generados al comienzo de la ejecución de un programa.

Capítulo 4

Método de trabajo

Todo proyecto de ingeniería necesita de una metodología para poder abordar de manera óptima los objetivos propuestos al comienzo de un proyecto. Por esto, el trabajo plasmado en este documento ha necesitado de una con la cual gestionar los diferentes hitos establecidos en etapas iniciales. Además es importante destacar los sistemas con los que se trabaja a fin de exponer la adecuación de éstos a los objetivos del trabajo. En este capítulo se describe, por un lado, el trabajo desarrollado para cumplir el objetivo principal del proyecto y, por otro lado, la metodología de desarrollo que se ha seguido como base para su gestión. Además, se detalla la planificación del proyecto y el conjunto de iteraciones empleadas, junto a los medios *hardware* y *software* utilizados.

4.1 Propuesta para mejorar el proceso del aprendizaje de la programación

Esta sección está dedicada a describir el planteamiento que se propone en este trabajo. Dicha sección se desglosa en diferentes puntos, en los cuales se cubren todos los aspectos que se han tenido en cuenta para abordar el desarrollo de la propuesta. En primer lugar, se presenta una visión general con la que despejar cualquier duda al lector sobre lo que se pretende hacer en este trabajo. En segundo lugar, se describe a alto nivel la arquitectura del sistema, explicando la relación que existe entre cada uno de los sistemas que la componen. En último lugar, se detalla la problemática encontrada y se expone la solución desarrollada, explicando las ventajas que ésta ofrece con respecto a otras soluciones estudiadas.

4.1.1 Visión general de la propuesta

Para introducir y poner en contexto al lector sobre lo que se pretende hacer en este trabajo conviene seguir un enfoque a alto nivel, según el cual se presenta sin entrar en detalles técnicos la funcionalidad y el flujo de interacción del usuario con el sistema. De esta forma, se ofrece una panorámica del sistema en su totalidad, la cual ayude a entender aquellos aspectos que no han quedado claros en un primer momento. En la Figura 4.1 se ofrece un esquema general del sistema.

La idea con la que se ha desarrollado este sistema es en base a dos líneas fundamentales: (i) mejorar el el proceso de aprendizaje de la programación a quienes están aprendiendo a

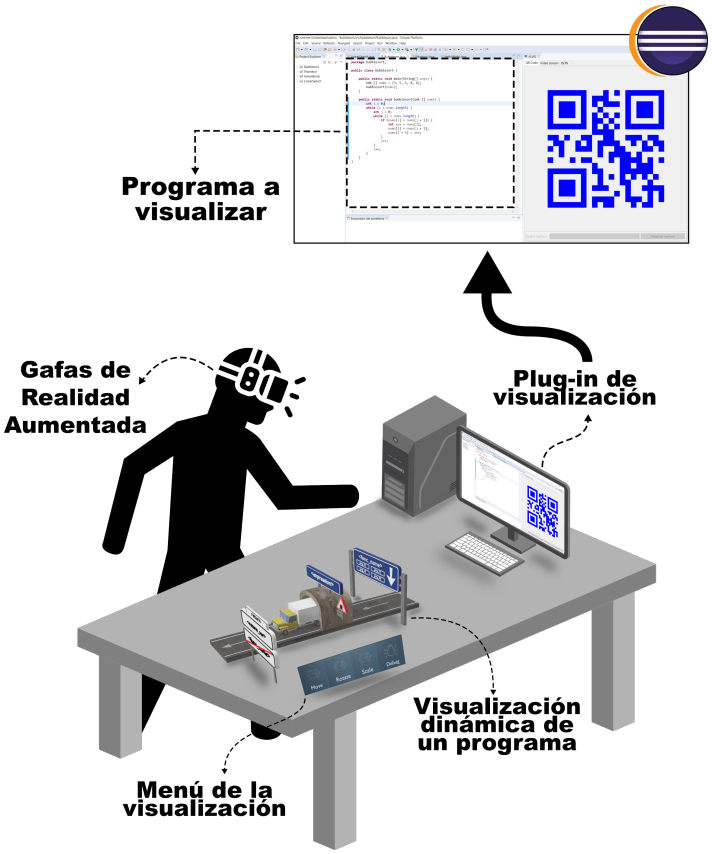


Figura 4.1: Perspectiva global del sistema

programar y quienes carecen de conocimientos para abordar problemas mediante un lenguaje de programación; y (ii) reforzar los conocimientos adquiridos de aquellos que tienen una experiencia previa en la programación.

Este sistema se apoya en una notación basada en una metáfora de carreteras y señales de tráfico para generar visualizaciones que representen el código fuente de un programa. Este enfoque pretende ser motivador y fácil de comprender, al incluir elementos familiares de la vida cotidiana de los estudiantes.

Dicha notación se establece como un conjunto de representaciones que mantienen una relación de correspondencia con sentencias de cualquier lenguaje, aunque se ha particularizado en este trabajo para el lenguaje Java. Estas sentencias son: definición de función, sentencia condicional, sentencia de bucle, evaluación de expresión y retorno de una función. Como ampliación a este conjunto se añade una representación para los puntos de ruptura y otra para simular el hilo de ejecución de un programa.

Estas visualizaciones son generadas automáticamente a partir del código fuente de un programa, según el orden de aparición de las sentencias. Esta generación se encuentra implementada como un *plug-in* en Eclipse, extrayendo la información relevante del código fuente para ser convertida a un formato (JSON) de intercambio de datos que es fácilmente interpretado por el dispositivo de RA. Este dispositivo se encarga de componer y mostrar las

representaciones gráficas tras el análisis, dando la opción, a través de un menú, de modificar su posición, rotación o tamaño.

Con esta propuesta, este sistema permite que usuarios puedan ver el código fuente de un programa desplegado de forma visual. Existe además la posibilidad de que profesor y alumnos, con varias gafas, puedan analizar ese código simultáneamente, comentando aspectos de interés durante el proceso de aprendizaje.

Esta visualización se consigue dirigiendo la mirada del usuario por la representación, quien tiene total libertad de movimiento al utilizar unas gafas que mantienen una correspondencia entre su posición y la posición de la visualización. Además, estas visualizaciones permiten representar la ejecución de un programa, usando la metáfora del vehículo con la que moverse por la representación simulando ejecutar sentencias. Esta funcionalidad se inicia mediante una nueva acción asociada al menú de la representación. Dicha acción manda a desplegar la herramienta de depuración de Eclipse, con la cual se establece una conexión para guiar al vehículo en función de la evaluación de las sentencias del programa.

A modo experimental, la funcionalidad descrita anteriormente da pie a poder visualizar varios procesos o hilos en ejecución, los cuales pueden ser seguidos de un modo personalizado por las características de la tecnología de visualización escogida.

En los siguientes puntos se describirá el enfoque utilizado para la construcción del entorno virtual aquí comentado, aportando todas las cuestiones ingenieriles que han permitido su implementación.

4.1.2 Metáfora de carreteras y señales de tráfico

Las representaciones de código fuente propuestas se plantean mediante una abstracción o metáfora basada en carreteras y señales de tráfico. Este conjunto permiten visualizar el flujo de ejecución de un programa de forma natural para el usuario, al encontrarse familiarizado con ellas en su vida diaria.

Cada una de estas representaciones se encuentran asociadas directamente a ciertas sentencias del lenguaje. El conjunto de sentencias a visualizar es lo suficientemente rico como para poder representar cualquier programa con ellas. No se hace distinción entre los tipos de bucles, como los clásicos *for*, *while* y *do-while*, al igual que con los tipos de condiciones *if-then-else* y *switch-case*, sino que todos ellos se engloban en una única representación, abstrayendo así al usuario de los detalles de implementación del lenguaje.

Previamente este conjunto permitía representar la estructura de los programas (visualización estática) [GP18]. El objetivo de la visualización estaba orientado a facilitar la comprensión del conjunto de sentencias que forman el programa. Esto era una gran limitación, al no tener la capacidad de analizar los fragmentos de código de forma dinámica al no poder ejecutarlos y depurarlos. Por tanto, el proceso de aprendizaje también estaba muy limitado. Por

ello, la metáfora ha sufrido una modificación y evolución, adaptando el conjunto de elementos gráficos a dicha metáfora y dando soporte con nuevos diseños a la visualización dinámica de programas. Por un lado, se ha considerado una alternativa para la metáfora de la expresión de variables que contemplara un diseño más adaptado e integrado con la misma. Como consecuencia, se optó por uno basado en un túnel de carreteras, donde la expresión que se evalúa es trasladada a una señal de tráfico sobre el propio túnel. Por otro lado, a modo de soportar la visualización dinámica de programas se han añadido dos representaciones cuyos diseños se adecúan perfectamente a la metáfora. Uno de ellos es el concepto de hilo, representado como un vehículo con forma de camión. El otro es un diseño con forma de cono de tráfico, el cual representa el concepto de punto de ruptura.

En consecuencia, este nuevo enfoque dota a la metáfora de una mayor versatilidad, cuya modificación y ampliación permiten que programas muestren tanto su aspecto estático como su aspecto dinámico. En la Figura 4.2 se presenta la notación en su totalidad tras las adaptaciones e incorporaciones anteriormente comentadas.

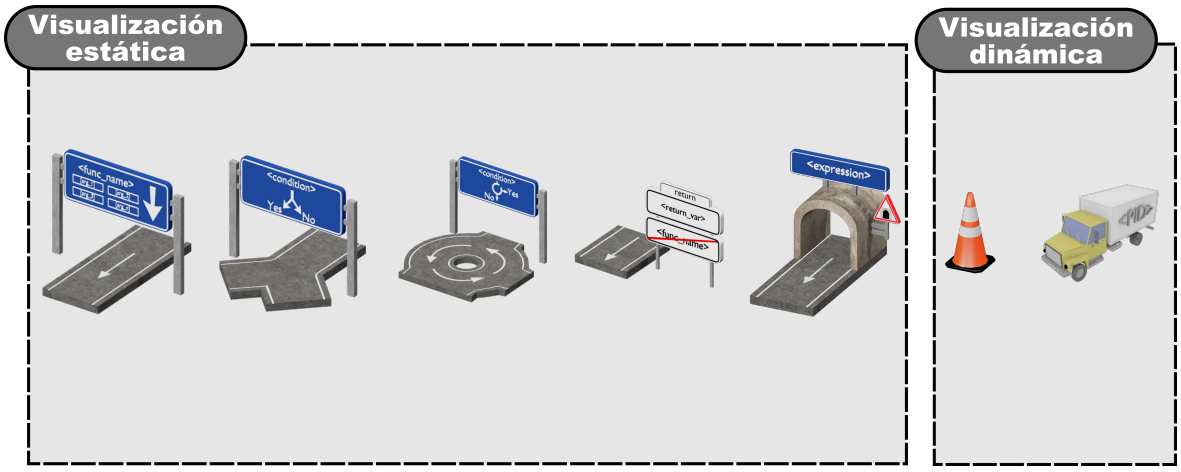


Figura 4.2: Nuevo conjunto de representaciones gráficas que componen la notación ANGELA. El conjunto de representaciones estáticas se utiliza tanto para la visualización estática como en la dinámica. Sin embargo, las metáforas para representar aspectos dinámicos son solo usadas para este propósito.

4.1.3 Arquitectura del sistema

Este trabajo se apoya sobre una arquitectura de red cliente-servidor, donde cada elemento está compuesto por un conjunto de módulos o subsistemas que desempeñan una serie de tareas específicas. La ventaja de este modelo es que cada subsistema es separado por una funcionalidad específica. De esta manera, el desarrollo se hace de forma organizada y se aíslan responsabilidades. Veamos a continuación el rol que desempeñan cliente y servidor, y cuáles son los subsistemas por los que éstos se componen.

- **Servidor**. Este elemento representa el núcleo del sistema, dado que se encarga de analizar y convertir el código Java a un formato fácilmente interpretable (JSON) por el cliente (HoloLens). Así, se consigue que la representación gráfica mantenga una correspondencia con el código fuente del programa a visualizar.
  - *Sistema de depuración*. Es el responsable de lanzar el depurador de Eclipse y de gestionar todo el proceso de depuración. Todo esto es realizado mediante el *plug-in org.eclipse.jdt.debug* que proporciona el entorno de Eclipse.
  - *Sistema de comunicación*. Se encarga de establecer y gestionar una comunicación entre el cliente y servidor. En otras palabras, este sistema es el responsable de establecer un puente entre el *plug-in* de visualización y el dispositivo Microsoft HoloLens. La comunicación entre ambas aplicaciones es a través de *sockets* de red TCP.
- **Cliente**. Recibe un archivo en formato JSON para generar y visualizar una representación gráfica compuesta por un conjunto de modelos basados en la metáfora de carreteras y señales de tráfico. La visualización puede representar el aspecto estático de un programa (estructuras de control) o el aspecto dinámico del mismo (ejecución de un programa).
  - *Sistema de visualización dinámica*. Se ocupa de generar una animación a partir de la evaluación de un programa. Es decir, guía a un vehículo sobre un programa con forma de carreteras ejecutando sentencias.
  - *Sistema de gestión de archivos* JSON. Se responsabiliza de gestionar la recepción y envío de archivos JSON para sincronizar correctamente la visualización y la depuración. Esto es, cuando el vehículo pretende pasar por una sentencia, el cliente avisa al servidor de que el depurador debe ejecutar una sentencia y así enviarle éste al cliente una respuesta con el valor de esa evaluación.
  - *Sistema de efectos audiovisuales*. Se encarga de gestionar efectos visuales y de sonido relacionados con la visualización dinámica.

En las siguientes secciones, se explican en detalle los subsistemas que forman parte de esta arquitectura, estructurando el documento en dos bloques diferenciales, servidor y cliente. A modo explicativo, la Figura 4.3 muestra un esquema que trata de mostrar la relación de todos los subsistemas del entorno, agrupados por las entidades descritas anteriormente.

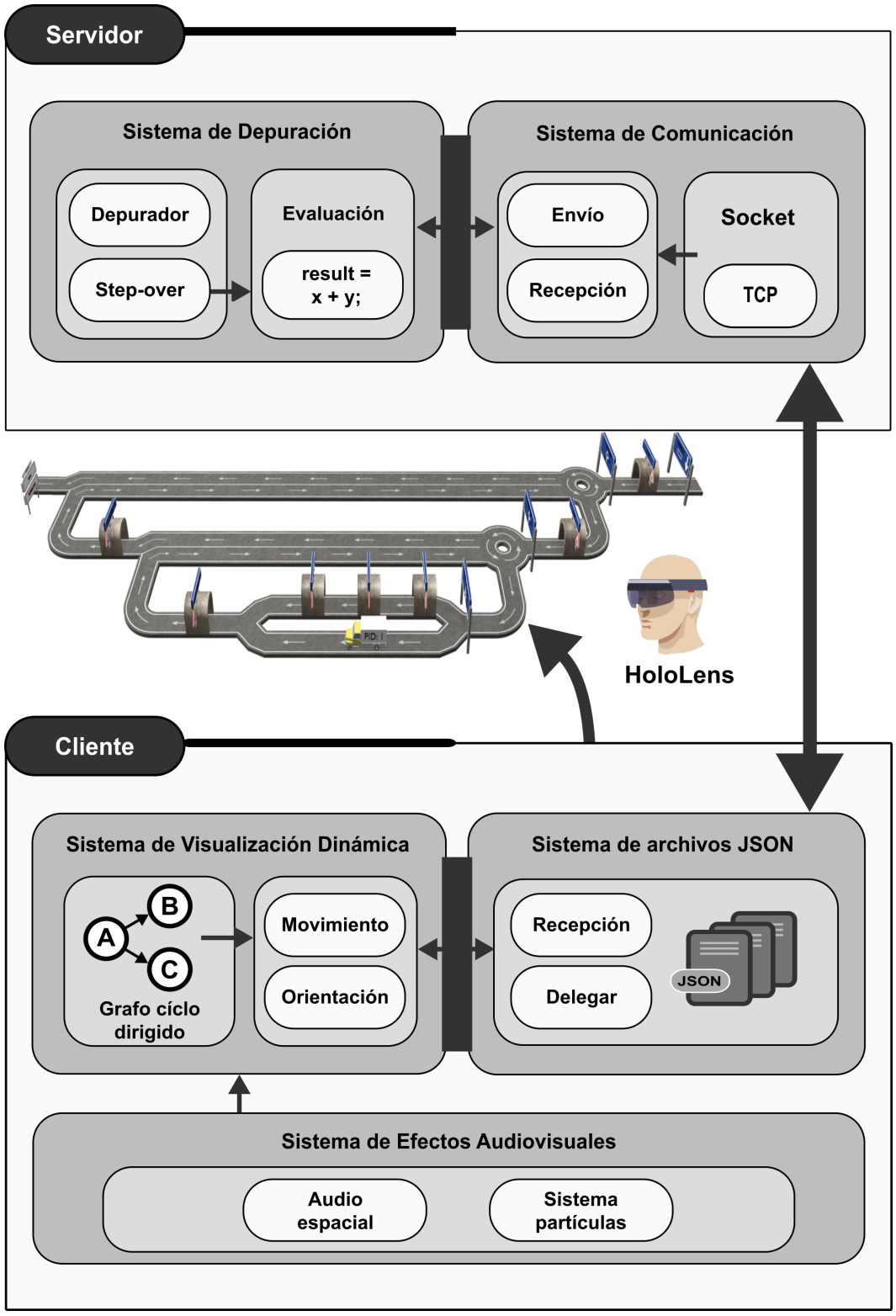


Figura 4.3: Arquitectura del sistema

4.1.4 Sistema de depuración

Este sistema es el encargado de desplegar las herramientas de depuración del entorno de Eclipse, así como gestionar la depuración de un programa mientras éste es visualizado dinámicamente por el dispositivo de RA.

La visualización dinámica de un programa consiste en representar, de forma animada, la ejecución de un programa. Esto consistía en obtener, para cada ejecución, el valor actualizado de cada variable. Para resolver esto surgieron dos enfoques: (1) utilizar programación orientado a aspectos o (2) desplegar el depurador de Eclipse de forma programática.

Ambos enfoques fueron estudiados detenidamente. Sin embargo, se optó por la idea de desplegar el depurador de Eclipse, dado que permite dotar al sistema de una mayor versatilidad, heredando funcionalidades como las de inspeccionar variables, poner puntos de ruptura, ejecutar programas paso a paso, etc.

Para resolver este planteamiento se ha hecho uso del *framework* JDT Debug[1], el cual consta de varios *plug-ins* para ejecutar y depurar programas en código Java. Éstos son: (1) **org.eclipse.jdt.launching**, (2) **org.eclipse.jdt.debug** y (3) **org.eclipse.jdt.debug.ui**. El primero permite lanzar la *Java Virtual Machine (*JVM*)* de forma programática. El segundo proporciona un conjunto de clases para acceder a objetos de un programa que está siendo depurado. El último concede acceso a las pestañas de la Interfaz de Usuario (IU) relacionadas con la depuración, por ejemplo, puntos de ruptura, variables, expresiones, etc.

La utilización de estos *plug-ins* han permitido crear una nueva funcionalidad que se apoya sobre una implementación desarrollada en [GP18] previo para analizar y procesar el código fuente de un procedimiento que un usuario quiere visualizar. Esto es así porque si no existe una visualización previa, tampoco existe una visualización dinámica de la misma. En otras palabras, para evaluar de una forma animada el código de un algoritmo se necesita en una primera instancia una representación de ese código. De tal manera que el usuario sobre esa representación pueda generar una visualización dinámica para comprender el comportamiento de dicho algoritmo.

La implementación que aquí se describe se apoya en varias clases ya diseñadas e implementadas, lo cual confirma la escalabilidad del sistema. La Figura 4.4 muestra un diagrama en el que se presenta la estructura de clases de este sistema. Los elementos remarcados indican elementos nuevos o modificaciones sobre elementos ya existentes.

En las siguientes secciones se describe el cometido de las clases involucradas en este módulo, además de explicar su construcción y la interacción del usuario con esta parte del sistema.

[1] `https://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Fguide%2Fjdt_int_debug.htm`

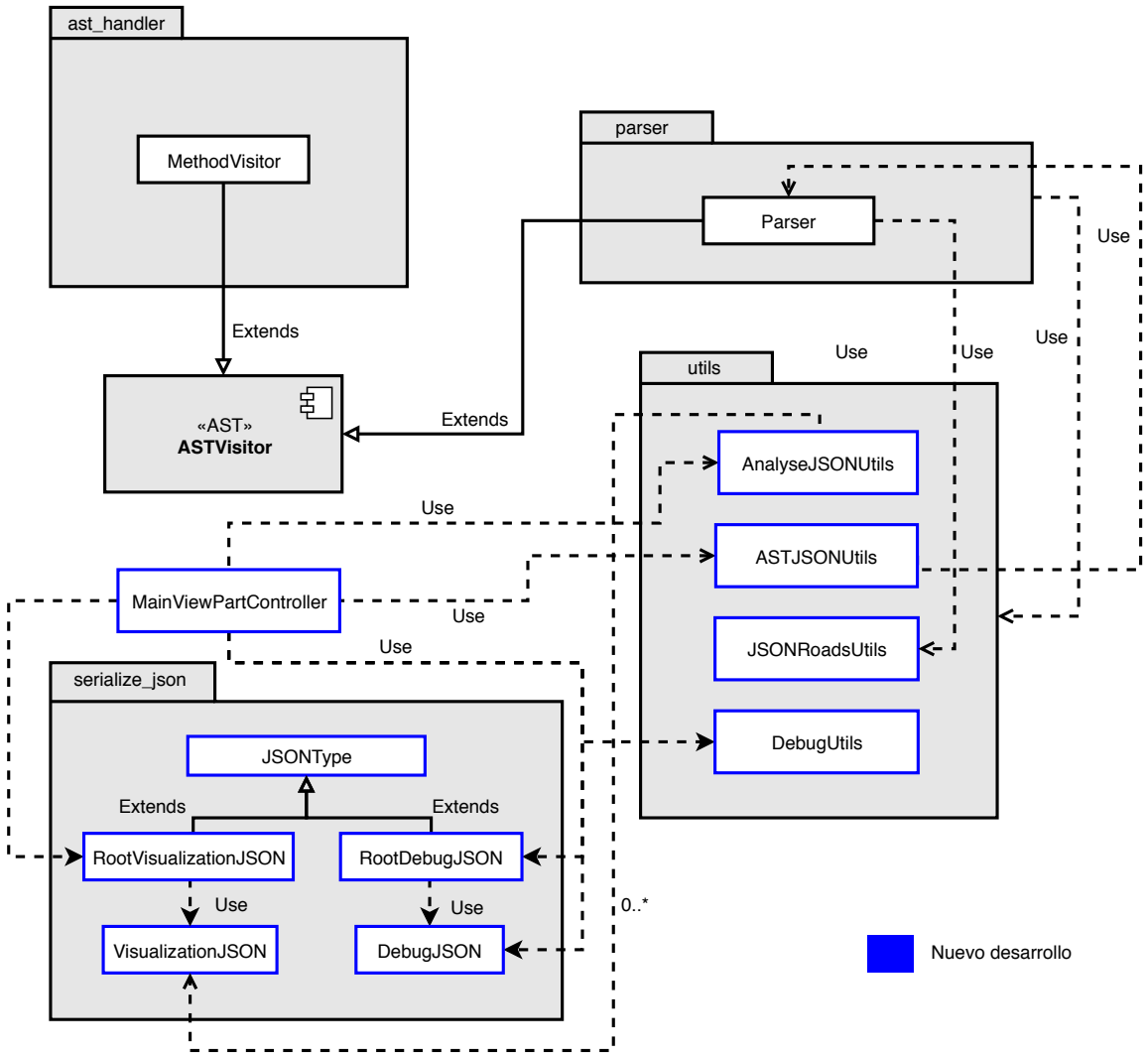


Figura 4.4: Elementos que componen el módulo de depuración

4.1.4.1 Despliegue del depurador

Ya se ha comentado que la arquitectura del sistema se ha pensado para que el usuario no realice una excesiva interacción con el sistema. De este modo, contribuimos a que el usuario centre toda su atención en la visualización del programa, proporcionándole un entorno configurado de un modo automático.

En primer lugar, y como se ha descrito anteriormente, la generación de una visualización dinámica parte de una previa visualización, es decir, el usuario en una primera instancia debe elegir un fragmento de código que desea visualizar para, posteriormente, producir una animación con dicha visualización de la evaluación de ese código.

Veamos primero cómo el usuario genera una visualización a partir de un algoritmo o programa, con el objetivo de poner en contexto al lector y basar esta explicación sobre esta generación. La Figura 4.5 refleja, a través de una serie de pasos, la interacción que realiza el usuario para generar una representación gráfica a partir de un fragmento de código. Partimos

del hecho que el usuario tiene descargado el *plug-in* de visualización y que las gafas de RA están conectadas a este entorno. El usuario tiene implementado un algoritmo en Eclipse y quiere visualizarlo para obtener una representación más natural. Para ello, se dirige a la vista del *plug-in* y selecciona el procedimiento a visualizar. Inmediatamente las gafas reciben un archivo, el cual es analizado para generar la conversión de código Java a una representación basada en una metáfora de carreteras y señales de tráfico.

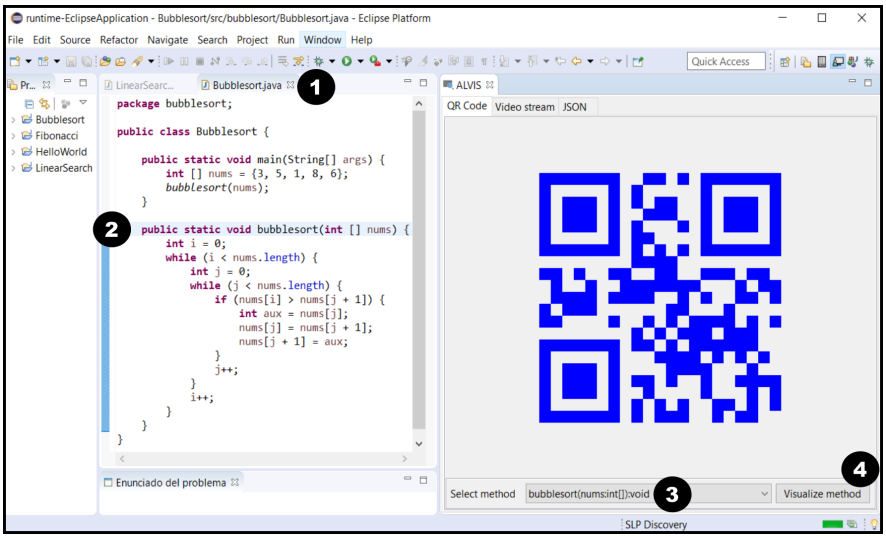


Figura 4.5: Interacción del usuario con el sistema para generar una visualización a partir de un fragmento de código: (1) clase que contiene o contendrá el procedimiento, (2) implementación del procedimiento, (3) selección del método a visualizar y (4) generación de visualización en el dispositivo de RA.

Esta representación tiene asociada un menú, con el cual se pueden realizar diferentes acciones: escalar, rotar, mover, eliminar o iniciar depuración. En esta sección centramos la atención en la última acción, al ser uno de los nuevos aportes de este trabajo. El usuario quiere realizar una visualización dinámica de la representación, por lo tanto, selecciona la acción de iniciar depuración. Este proceso genera, de forma transparente, una estructura en formato JSON, la cual es enviada al servidor mediante *socket* de red TCP para desplegar el depurador en Eclipse. Esta estructura mantiene información sobre el método a depurar y el modo de la depuración (ver Listado 4.1). Básicamente, lo que se pretende con esta estructura es automatizar el despliegue del depurador (*start*), la ejecución paso a paso (*step over*) de un programa o la finalización del mismo (*finish*).

Una vez que esta estructura llega en forma de mensaje al servidor, un manejador es el encargado de comprobar su propósito, el cual viene definido al comienzo del archivo. Tras esta comprobación, el manejador ordena que se cree un punto de ruptura en la línea del código fuente que declara la signatura del método. Este procedimiento se encuentra definido en la clase *DebugUtils*, la cual se encarga de mantener todos lo métodos necesarios para trabajar con el depurador de Eclipse. Es importante destacar que el método a depurar debe

```
{
  "type":"debug",
  "debug": [{
          "method":"method-name",
          "mode":"start|step-over|finish"
      },
      ...
]}
```

Listado 4.1: Estructura JSON para la gestión de la depuración

ser definido en el método principal del programa (*main*). De lo contrario, el depurador no se desplegará y no se realizará la visualización. A continuación, se configuran los parámetros del lanzador, los cuales son guardados localmente en los metadatos del *workspace*[2]. Tras esto, se localiza y se obtiene la referencia de la pila de la JVM suspendida a causa de un punto de interrupción. La necesidad de obtener esta referencia es por el hecho de que almacena el contexto de la ejecución. Por lo tanto, a través dicha referencia, se accede a las variables que están dentro del ámbito del método que está siendo depurado. A modo de ejemplo, en la Figura 4.6, se representa el flujo usado para el despliegue del depurador.

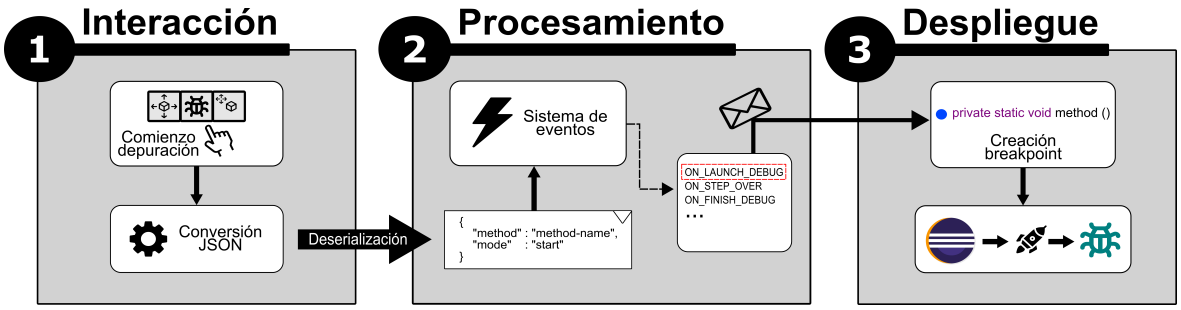


Figura 4.6: Proceso para desplegar el depurador de Eclipse

4.1.4.2 Gestión de la ejecución paso a paso

Una vez el depurador ha sido desplegado, el *plug-in* se mantiene a la espera de recibir una orden en formato JSON. Estas órdenes son enviadas desde el dispositivo de RA, las cuales son emitidas una vez el vehículo atraviesa una sentencia en forma de representación gráfica.

Anteriormente se ha comentado la definición de una estructura JSON para mantener información sobre la depuración de un programa. Esta información viene representada por el nombre del método que se quiere depurar y un modo (lanzamiento, paso a paso o finalización). Las razones de la elección de esta notación se deben a que proporciona un formato de texto sencillo, fácilmente interpretable y soportado por varios lenguajes de programación.

Para el caso que estamos describiendo, el modo que espera el servidor recibir es la acción

[2]Directorio en el que se agrupan proyectos, sus configuraciones y los ajustes del editor.

de ejecutar paso a paso un programa (o *step over* en inglés), cuya orden fuerza al depurador a pasar sobre la actual sentencia contenida en la pila del hilo suspendido. Afortunadamente, las estructuras de hilo, pila, contexto de depuración, variables y valores son proporcionadas por el entorno de Eclipse mediante el *plug-in org.eclipse.jdt.debug*, los cuales facilitan enormemente la gestión de un proceso de depuración (ver Figura 4.7).

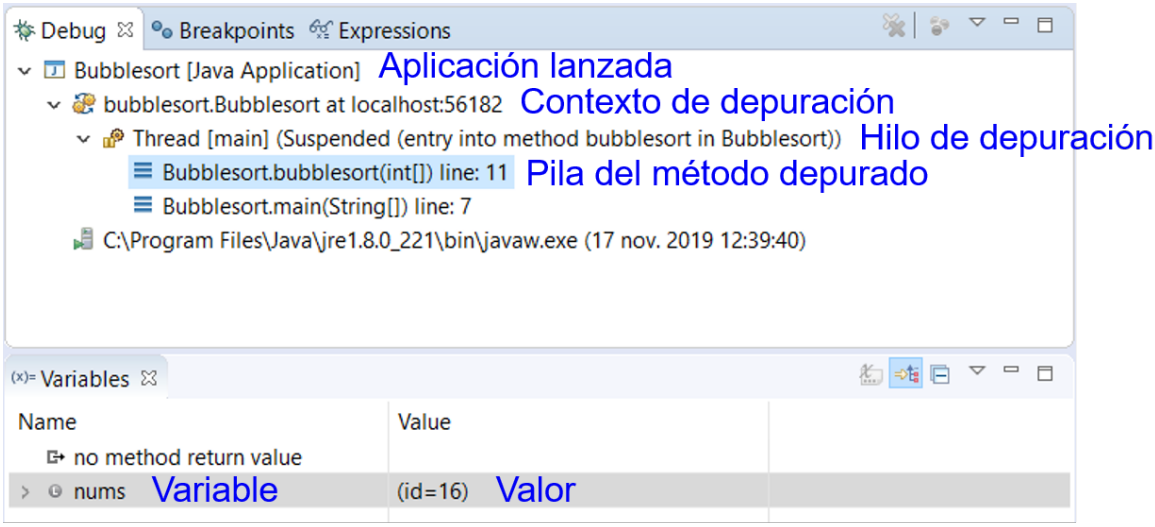


Figura 4.7: Perspectiva de depuración en Eclipse. Estructuras involucradas en el proceso de depuración.

Sin embargo, existe la necesidad imperiosa de sincronizar toda esta depuración con la parte gráfica. Se ha descrito en secciones anteriores que el vehículo debe circular por la metáfora en función de la evaluación retornada por el depurador. Por ejemplo, en el caso en el que el vehículo se encuentre con una sentencia condicional, representado en la visualización mediante una bifurcación, el nivel de sincronización entre servidor y cliente será crucial, dado que dicho elemento seguirá un camino u otro, condicionando completamente la evaluación del programa. Este proceso será descrito más en detalle en secciones posteriores.

La ejecución de la acción *step over* se realiza para obtener el valor de la sentencia que estamos evaluando. Esta evaluación se puede conseguir una vez que la acción se ha completado correctamente. Para asegurar esto, el hilo de depuración debe ser suspendido una pequeña porción de tiempo (50 milisegundos como tiempo mínimo) para que la JVM cuente con el tiempo suficiente para actualizar la pila de variables. Tras esto, aseguramos que la evaluación de la expresión esté lista, integrando el valor en la estructura JSON para ser enviada al dispositivo de visualización.

4.1.5 Sistema de comunicación

Este sistema es el encargado de coordinar y sincronizar la comunicación entre el cliente y servidor utilizando *sockets* de red TCP. La primera versión de este sistema pertenece al desarrollo que se realizó en [GP18]. A modo de resumen, las características de este sistema

eran: envío de archivos JSON al cliente con la conversión del código Java y la detección de la conexión del dispositivo de RA.

La funcionalidad de este sistema, para el cometido propuesto, era suficiente. Sin embargo, para mantener una sincronización en tiempo real entre *plug-in* y dispositivo de visualización es necesario una mejora sustancial. Este sistema trata de dar una solución a dos problemas: (1) la gestión de las peticiones del cliente y (2) el problema a la sincronización sin errores entre cliente y servidor.

El primer problema surge porque este sistema debe ser capaz de gestionar diferentes tareas, lo cual siempre dificulta en cierta medida la implementación. Una de estas tareas es la gestión de la sincronización del dispositivo de visualización con el *plug-in*. Esto es uno de los mayores problemas a los que este sistema se enfrenta, dado que durante la visualización dinámica un mensaje puede perderse. Esto puede suponer que la visualización no mantenga una correspondencia con la evaluación, por lo tanto, el objetivo que se intenta conseguir con este proceso carecerá de valor para el estudiante. Aparte, el resultado de la evaluación del depurador debe ser enviada a tiempo al cliente, con el objetivo que el vehículo pueda moverse correctamente por la representación. Por otro lado, el sistema debe controlar qué tipo de acción se necesita llevar a cabo, por ejemplo, si es necesario desplegar el depurador, si éste tiene que evaluar una sentencia o si debe terminar este proceso.

Con respecto al primer caso, una de las soluciones adoptadas ha sido la utilización de *sockets* de red TCP. El empleo del protocolo TCP se debe a que la comunicación debe ser segura y fiable. Esto significa que se asegura el envío y recepción de paquetes. Por lo tanto, se garantiza que nunca haya pérdida de información y, por consiguiente, que la visualización se espera que se realize con total normalidad. Su elección se debe a que no puede existir fallo durante la visualización, ya que existe un intercambio de información entre cliente y servidor para mantener una sincronización entre el depurador y la visualización. Además, se ha integrado la biblioteca *netty*[3], un componente que ofrece una capa de comunicación transparente al desarrollador.

En cuanto a la gestión de las peticiones del cliente, esto se ha solventado mediante la construcción de una estructura en formato JSON, la cual fue descrita en la sección anterior. El sistema a partir de la recepción de un archivo JSON identifica la acción que éste debe realizar, actuando como una especie de manejador. En función de dicha información, el manejador notifica al controlador de la vista para que ejecuta la acción pertinente. En el Listado 4.2 se muestra el procedimiento encargado de identificar el propósito del archivo JSON y notificar a la vista principal que realice una acción.

El diseño empleado en este módulo ha ofrecido ciertas ventajas al sistema. Por un lado, la utilización de JSON ha permitido facilitar la gestión de las tareas relacionadas con la depu-

[3] `https://netty.io/`

```
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        ...
        switch(mode){
        case ConstantJSON.START:
                String methodName = debugJSON.getMethod();
                Display.getDefault().asyncExec(() -> {
                        ViewPartManager.INSTANCE.notifyAllViews(
                                MainViewPart.Event.ON_LAUNCH_DEBUG, methodName);
                });
                break;
        case ConstantJSON.STEP_OVER:
                Display.getDefault().asyncExec(() -> {
                        ViewPartManager.INSTANCE.notifyAllViews(
                                MainViewPart.Event.ON_STEP_OVER, json);
                });
                break;
        case ConstantJSON.FINISH:
                Display.getDefault().asyncExec(() -> {
                        ViewPartManager.INSTANCE.notifyAllViews(
                                MainViewPart.Event.ON_FINISH_DEBUG);
                });
        }
}
```

Listado 4.2: Método que gestiona la recepción de un mensaje del cliente

ración. Este formato permite desacoplar el código del cliente y servidor, de tal forma que un cambio en uno de estos dos elementos no suponga la modificación del código del otro. Por el contrario, la integración de *netty* y TCP han supuesto un punto de inflexión en el desarrollo. La biblioteca *netty* ha facilitado la recepción y envío de mensajes, mientras que la elección de TCP ha servido para mantener un sistema robusto y fiable en cuanto a comunicación se refiere, evitando pérdida de información importante.

4.1.6 Sistema de visualización dinámica

Este sistema es el encargado de realizar visualizaciones dinámicas de un programa a partir de una metáfora de carreteras y señales de tráfico. Dado el código fuente de un algoritmo y su conversión a un formato JSON, este sistema lo traduce a una representación gráfica compuesta por modelos que siguen la filosofía de la metáfora propuesta, con el objetivo de que un vehículo circule por esta estructura simulando ejecutar sentencias.

Este planteamiento conllevaba tener en cuenta dos aspectos importantes: (1) cómo mover el vehículo siguiendo un recorrido y (2) cómo hacer que ese recorrido sea en función de la evaluación de las sentencias de un programa.

Para solventar la primera cuestión se necesitó implementar una estructura que permitiese al vehículo conocer el siguiente punto al que tenía que llegar. Respecto a la otra, el sistema se

apoyó en el depurador de Eclipse para conocer la evaluación de cada sentencia y así moverse correctamente por la metáfora.

Estas soluciones se originan a partir de una jerarquía de clases que establecen una serie de relaciones. Con el objetivo de comprender dichas relaciones y la estructura de este módulo, la Figura 4.8 refleja en forma de diagrama el conjunto de clases que componen este módulo.

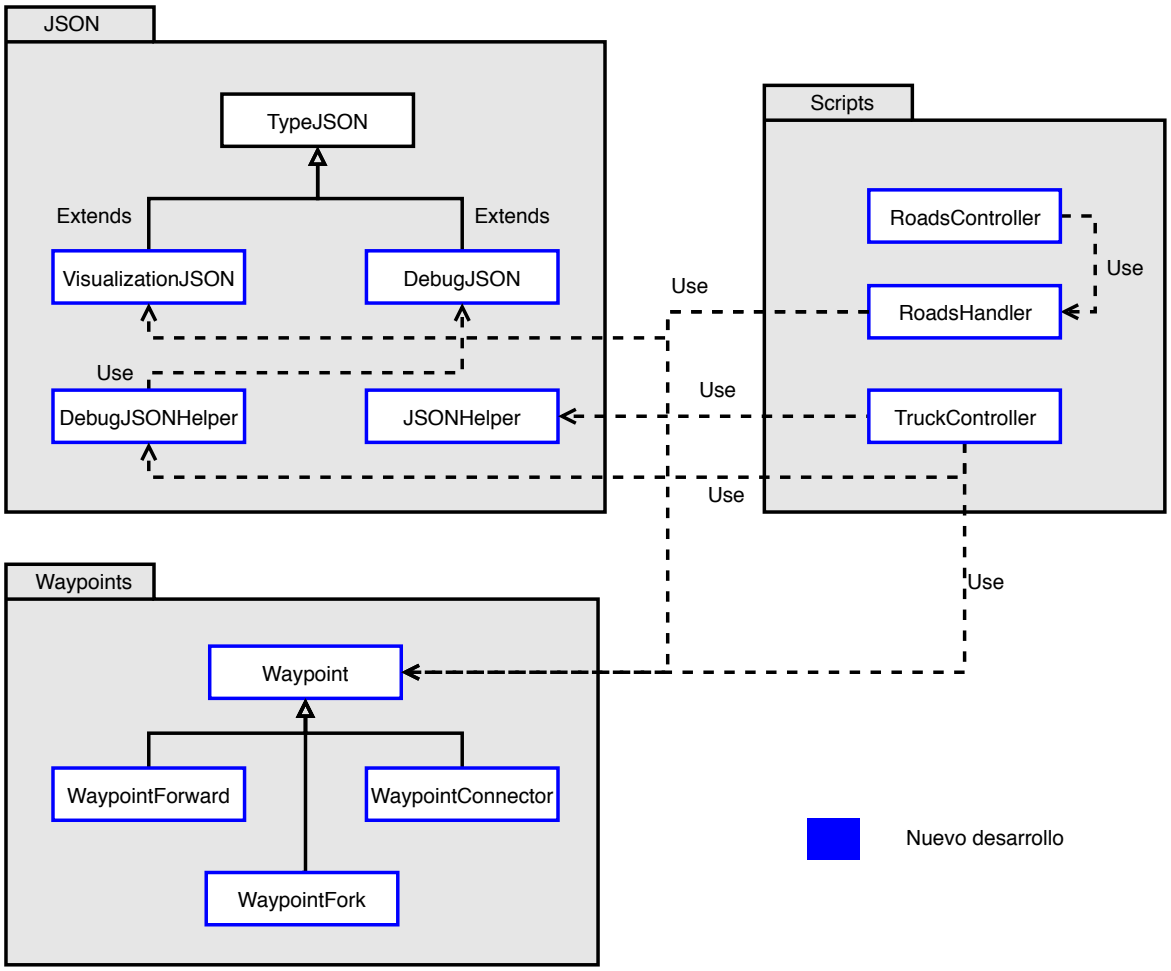


Figura 4.8: Elementos que componen el módulo de visualización

En las siguientes secciones se detalla en profundidad el desarrollo de cada uno de los enfoques anteriores, explicando su construcción y las ventajas que aportan.

4.1.6.1 Construcción dinámica de grafo cíclico dirigido

El principal y claro problema del módulo de visualización dinámica pasa por cómo conseguir que el vehículo siga un recorrido en función de la evaluación de las sentencias de un programa que a priori no se conoce. Es decir, el sistema no carga algoritmos previamente introducidos a modo de ejemplo, sino que el usuario tiene la libertad de elegir y elaborar sus propios programas para después ser visualizados en el sistema. Por lo tanto, éste debe ser lo suficientemente flexible para poder generar una visualización dinámica a partir de cualquier fragmento de código.

La primera idea con la que se ha pretendido resolver el problema ha sido utilizando el enfoque de la teoría de grafos. Un grafo consiste en un conjunto de nodos o vértices conectados por enlaces llamados aristas o arcos que representan relaciones binarias entre elementos de un conjunto. En este contexto, los nodos representan los puntos hacia los que el vehículo se tiene que dirigir y los enlaces son tramos de carreteras por los que éste debe circular. Así, se mantiene una estructura totalmente conectada para facilitar la dirección que debe tomar el vehículo. En esencia, un grafo representa conexiones. Sin embargo, y para concretar el enfoque, dependiendo del tipo de grafo, las conexiones formarán o no caminos cerrados. En este sentido, se eligió utilizar un *grafo cíclico dirigido*, el cual representa tanto caminos abiertos como cerrados, ya que el vehículo puede realizar recorridos en bucle.

Los nodos son una especie de puntos clave o *waypoints* compuestos por medio de esferas de colisión, un tipo de componente invisible que define la forma de una esfera a efectos de colisiones físicas. Estos componentes suelen asociarse a una malla con una forma similar al componente de colisión, para que cuando dicho objeto colisione, el movimiento o la física realizada sea similar a la que un objeto de esta forma tendría en el mundo real. No obstante, para este enfoque no es necesario el uso de una malla, dado que este componente de colisión es usado para detectar colisiones con la metáfora del vehículo. Así se consigue guiar y dirigir a este elemento por la representación compuesta por carreteras.

Este tipo de componente está integrado en *Unity 3D*[4], entorno utilizado para el desarrollo gráfico de este sistema. *Unity 3D*, aparte de ser un motor de juegos, es también utilizado para el desarrollo de aplicaciones gráficas. En este motor, el desarrollo se realiza utilizando el *framework* .NET[5] de Microsoft, cuyo código después es convertido en función de la plataforma de destino (p.ej., *windows*, *ios*, *android*, etc.) en la que se quiere ejecutar el producto desarrollado.

Unity trabaja con *GameObjects*, un tipo de objeto al que se le puede asociar tantos componentes como se quiera. En nuestro caso, los nodos son *GameObjects*, los cuales tienen como componente asociado una esfera de colisión, es decir, un *sphere collider*.

En primer lugar, este objeto muestra un componente de tipo *Transform*, el cual es asociado automáticamente en todos los *GameObjects*. Este componente mantiene información sobre la posición, rotación y escalado de nuestro objeto. Aparte, este objeto también cuenta con un componente de tipo *sphere collider*, el cual define una esfera de colisión. Aparte, este mismo componente tiene seleccionado el campo *IsTrigger*, el cual permite que se dispare un evento cuando un objeto colisione con éste.

Estos puntos clave son generados al mismo tiempo que se compone la representación gráfica, es decir, el momento en el que se unen los modelos 3D para dar forma a una representación gráfica unificada. Así, se realiza toda la generación en un solo paso, evitando

[4] `https://unity3d.com/es`

[5] `https://docs.microsoft.com/es-es/dotnet/`

costes computacionales adicionales.

Dependiendo de la representación construida, el punto clave a generar cambia. Como se ha visto en el diagrama de clases de este sistema, hay tres tipos de puntos clave o *waypoints*, los cuales son: *waypointForward*, *waypointConnector*, *waypointFork*. El primero define puntos clave que solo tiene un único camino y que mantienen la información de una sentencia del programa que se visualiza. En otras palabras, indica que el movimiento del vehículo es solo hacia adelante y que se encuentra en un nodo que mantiene información sobre una sentencia del código fuente. El segundo es un nodo que actúa en forma de conector, es decir, no se mantiene información sobre ninguna sentencia, sino que solo sirve para poder guiar al vehículo por curvas o giros complejos. Por último, el punto clave *waypointFork* sirve para definir un nodo que tiene dos posibles caminos, el cual es usado con la representación en forma de bifurcación y en otros puntos clave. A través de esta implementación, el nodo mantiene una conexión con los dos siguientes caminos, de tal manera que, en función del resultado retornado por el servidor, el vehículo podrá seguir con su recorrido sin ningún problema.

Para comprender mejor lo que se describe en este apartado, veamos un ejemplo de los pasos que el sistema utiliza para generar un grafo. Imaginemos que el usuario quiere visualizar el algoritmo de la burbuja (*bubblesort*). Para ello, partimos del hecho que el dispositivo de visualización y el servidor están conectados, con el objetivo de no extender demasiado el ejemplo. A continuación se muestra en forma de lista enumerada el conjunto de acciones involucradas.

1. El usuario inicia el proceso de visualizar el algoritmo a través de las gafas de RA.
2. El *plug-in* de Eclipse convierte el código Java a un formato en código JSON.
3. Este archivo es recibido por las gafas, las cuales interpretan dicho archivo para generar una representación gráfica unificada.
4. La representación se genera según el orden de aparición de las sentencias en el lenguaje Java.
5. Cada vez que se convierte una sentencia en una representación gráfica (p.ej., expresión en túnel con carretera, definición de función en señal de tráfico con carretera, etc.), una serie de esferas de colisión son creadas sobre la superficie de esa metáfora. Así, se consigue tener un conjunto de esferas que componen un grafo, el cual sirve de ayuda para poder mover el vehículo por la representación.

En la Figura 4.9 se simplifica la explicación anterior para dar pie a una mejor comprensión. En la parte izquierda de la imagen, se añade el código del algoritmo seleccionado a modo de ejemplo. A la derecha, el algoritmo convertido en la metáfora de carreteras con las esferas de colisión y unos identificadores que reflejan la correspondencia entre las sentencias del código y las representaciones gráficas. Además, se muestra de manera independiente el grafo generado para este algoritmo. Es importante resaltar que estas esferas no se visualizan

durante la ejecución, sino que aparecen invisibles al usuario, siendo utilizadas para disparar eventos cuando el vehículo pase sobre ellas.

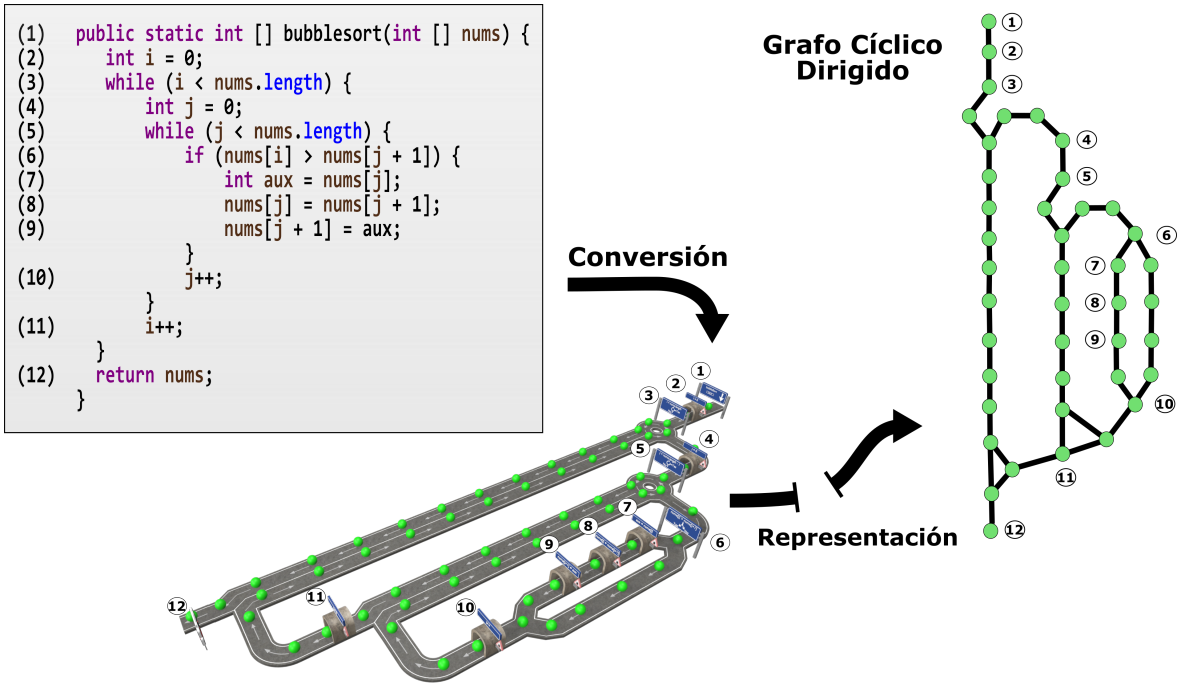


Figura 4.9: Ejemplo de la generación de un grafo cíclico dirigido para el algoritmo la burbuja. Los elementos verdes son las esferas de colisión que representan un nodo.

4.1.6.2 Movimiento del vehículo que representa un hilo en ejecución

Los hilos son representados por vehículos que circulan sobre la metáfora de carreteras y señales de tráfico. Esta circulación es generada en tiempo de ejecución, en función de la representación y de la evaluación de las sentencias. Este planteamiento genera una serie de cuestiones, como por ejemplo, ¿cómo se le da movimiento al vehículo?, ¿a qué velocidad viaja este objeto? o ¿cómo el sistema evita que el vehículo se salga de la representación? Cada una de estas cuestiones han sido abordadas en este trabajo. Sin embargo, la solución aportada no ha sido trivial. A continuación, se describe de forma detallada las soluciones aportadas, proporcionando material gráfico que facilite la explicación.

Para resolver el enfoque del movimiento del vehículo se ha hecho uso de varios métodos muy populares en el ámbito de los Gráficos por Computador. Por un lado, se han utilizado operaciones basadas en vectores y, por otro, cálculos con cuaterniones.

Antes de entrar en detalle en cómo se ha calculado la orientación del vehículo en función de la posición de un nodo, pasemos a ver la solución para impulsar el camión sobre la representación. En primer lugar, al vehículo se le debe dar una velocidad para poder moverse sobre las carreteras. Para ello, se ha añadido un componente al vehículo denominado *rigid-*

body[6]. Este componente permite al objeto que lo lleva actuar bajo el control de las físicas, pudiendo hacer que éste se mueva de manera realista.

Tras esto es importante determinar su orientación, ya que todos los nodos no estarán frente a éste. Para ello se ha hecho uso de vectores, con los cuales es fácil determinar la orientación de un objeto. Esto se consigue mediante la resta del vector que representa la posición del nodo objetivo ($\vec{B}$) y la posición actual del vehículo ($\vec{A}$). Es decir:

$$\vec{AB} = \vec{B} - \vec{A}$$

Una vez conocida la orientación se utiliza la función *LookRotation* de la clase *Quaternion*, la cual crea una rotación basada en las direcciones *forward* (vector que define la dirección a la que mira el objeto) y *upwards* (vector que define la dirección hacia arriba del objeto), es decir, crea una rotación que alinea el objeto hacia la dirección previamente calculada. En otras palabras, el vehículo en cada *tick* de reloj modifica su rotación para alinearse éste con el objeto hacia el que se dirige. De esta manera se evita que el vehículo salga del circuito y siga correctamente la dirección objetivo. Por último, para que la rotación no sea brusca se hace uso de la función *Slerp*, la cual es la equivalente a la función *Lerp* para rotaciones. Esta función interpola en forma de curva dos rotaciones dado un valor, es decir, dado un vector $\vec{a}$, un vector $\vec{b}$ y una cantidad de tiempo `t`, el vector retornado es la interpolación esférica entre los dos vectores dados. Véase la función de interpolación esférica[7] para más información.

4.1.7 Sistema de gestión de archivos JSON

Este sistema es el encargado de recibir y gestionar los archivos que el *sistema de comunicación* envía al cliente. El servidor genera archivos de intercambio de datos, los cuales se obtienen de analizar y procesar el código fuente de Java. Además, éste envía información acerca de la evaluación de las sentencias, donde el cliente es quien se encarga de notificárselo al vehículo para tomar una u otra dirección.

Como se puede apreciar, el servidor genera dos tipos de archivos JSON: uno para convertir el código Java a una representación de carreteras y señales de tráfico y otro con la evaluación de una sentencia previamente enviada por el cliente, es decir, el resultado que necesita el vehículo para moverse por la representación. Esto indica que es necesario disponer de un manejador que identifique el propósito del mensaje para delegar la tarea.

La solución adoptada pasa por indicar en el propio archivo el propósito del mensaje, el cual puede ser: *debug* o *visualization*. Al igual que en el sistema de depuración, un manejador recibe el archivo, identificando el tipo de mensaje para delegar la tarea en la clase correspondiente. Así, se evita tener el código desacoplado, lo cual permite tener una imple-

[6] `https://docs.unity3d.com/es/current/Manual/class-Rigidbody.html`
[7] `https://en.wikipedia.org/wiki/Slerp`

mentación escalable, de tal forma que los cambios realizados en la estructura del archivo JSON no repercutan en toda la implementación del sistema.

Aparte, la sincronización entre cliente y servidor es importante por el tipo de sistema desarrollado. Ambos necesitan establecer una conexión robusta, con el objetivo de que ésta perdure durante la ejecución del sistema. Debido al diseño del sistema, mantener la conexión es imprescindible para que el cliente conozca el resultado de la evaluación de cualquier sentencia. Esto es así porque cuando el usuario está realizando la visualización dinámica de un algoritmo el vehículo debe saber cuál es el resultado de una expresión o sentencia para elegir el camino correcto. Veamos esto con un ejemplo (ver Figura 4.10).

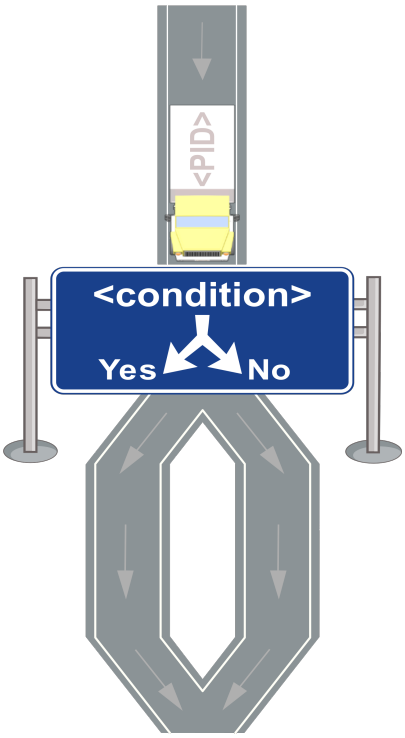


Figura 4.10: Situación de un vehículo apunto de entrar en una sentencia condicional

En la representación, se muestra un vehículo a punto de entrar en una bifurcación, la cual representa una sentencia condicional. En ese instante, el cliente envía la expresión de la sentencia condicional al servidor, quien se encarga de realizar la evaluación. Una vez obtenido un resultado, éste es enviado de vuelta al cliente, para que el vehículo en función de ese valor tome un camino u otro. Se puede apreciar que la recepción a tiempo de este mensaje es crucial, ya que puede desencadenar en la visualización errónea de un algoritmo.

Por esto, el protocolo de comunicación escogido es TCP, dado que es un protocolo que proporciona un transporte fiable de flujo de información entre aplicaciones. Además, está pensado para poder enviar grandes cantidades de información de forma fiable, liberando al programador la dificultad de gestionar la fiabilidad de la conexión (retransmisiones, pérdida de paquetes, orden en el que llegan los paquetes, duplicidad de paquetes, etc.). Estas ven-

tajas hacen descartar la posibilidad de utilizar User Datagram Protocol (UDP), dado que no proporciona una alta fiabilidad en cuanto al intercambio de información.

Para dar claridad a este planteamiento, el diagrama de secuencia mostrado en la Figura 4.11 refleja claramente esta sincronización, proporcionando el paso de mensajes que existe entre ambos elementos.

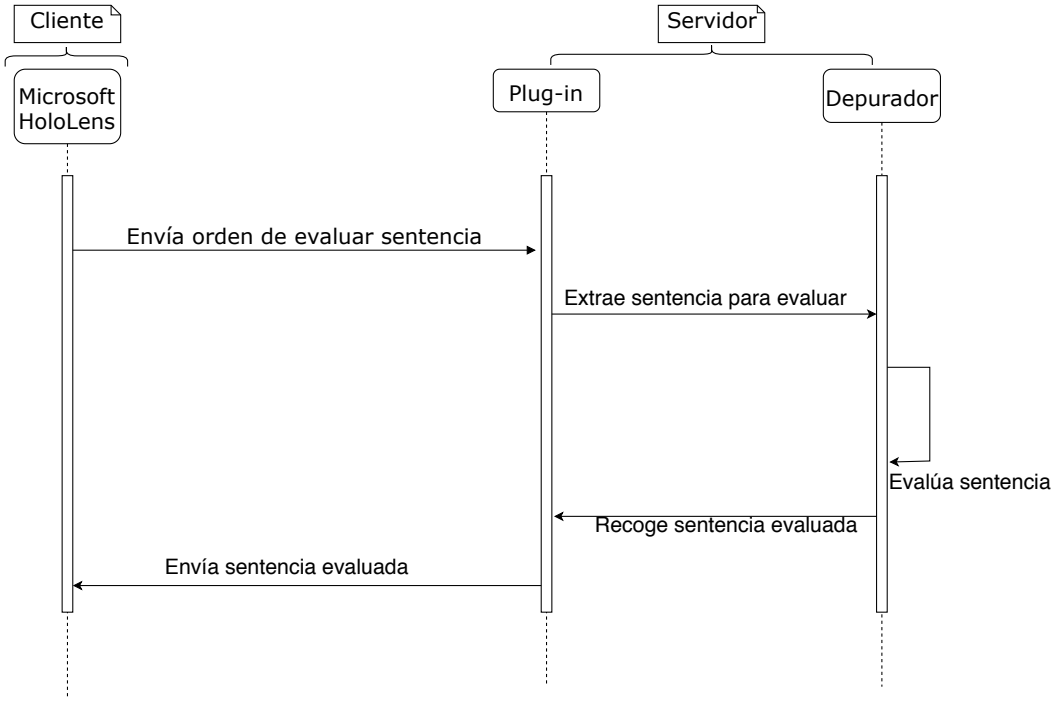


Figura 4.11: Intercambio de mensajes durante la visualización dinámica de un programa

4.1.8 Sistema de efectos audiovisuales

A efectos de proporcionar una experiencia de usuario diferente, este sistema proporciona partículas visuales y sonidos que tratan de sumergir aún más al usuario en un entorno de RA. Durante la visualización, dinámica este sistema dota al vehículo de partículas de humo que simulan el efecto de los gases de combustión que escapan del motor.

Las partículas comienzan a emitirse una vez el vehículo aparece sobre la representación y perduran hasta que la visualización dinámica finaliza. Esto es controlado desde un *script* asociado al manejador de la visualización, es decir, la clase que se encarga de gestionar la visualización. Además, junto a esta funcionalidad, el sistema lanza un sonido que simula el arranque y apagado del motor.

El primer método se ejecuta justo cuando el vehículo es creado. *StartCoroutine* es una llamada de la *Application Programming Interface (*API*)* de *Unity* que ejecuta un acción pasado un tiempo determinado. En este caso, el tiempo indicado es el que tarda el audio en lanzarse. Tras esto, el cuerpo de esta función se ejecuta, emitiendo las partículas de humo.

Por otro lado, en el momento en el que el vehículo es eliminado se activa el método *OnDisable*. Aquí, se detienen tanto las partículas como el audio del motor, liberando recursos del sistema.

Respecto al sonido producido para simular el motor de un vehículo, el sistema trata de reproducirlo de un modo espacial, es decir, dependiendo de la posición en la que el usuario se encuentre el sonido se reproduce con mayor intensidad en un oído o en otro, tratando de sumergir al usuario en un entorno de RA.

Para ello es necesario indicar en el componente de reproducción de audios que esta funcionalidad se active, dado que previamente aparece desactivada. Esta funcionalidad viene implementada por el motor. Sin embargo, su activación no es del todo trivial, ya que es fundamental saber cuál es el propósito de cada uno de los atributos que proporciona.

Los dos componentes imprescindibles para que el sonido espacial funcione son *spatialize* y *spatial blend*. El primero activa la función de que el sonido sea espacial, como bien lo indica su nombre. El segundo indica que el sonido sea en 3D, lo que significa que la intensidad del sonido aumente o disminuya en función de la distancia del usuario respecto al objeto que emite el sonido.

4.2 Planificación

De acuerdo a la metodología escogida, el proyecto se ha planificado mediante Paquetes de Trabajo (PT en adelante) los cuales mantiene una correspondencia con los objetivos marcados en el Capítulo 2. A su vez, estos PT están constituidos por varias Tareas (T en adelante), las cuales definen procesos específicos para completar un objetivo en concreto. Desde un punto de vista general, esta planificación se compone de cuatro PT, que son:

- **PT1. Propuesta de una metáfora de carreteras y señales de tráfico para soportar la visualización dinámica de programas**. Este paquete engloba una serie tareas para extender el sistema a la visualización dinámica de programas.
- **PT2. Diseño y codificación de un sistema de visualización dinámica**. Este paquete contiene las tareas relacionadas con el desarrollo del módulo de la visualización dinámica de programas.
- **PT3. Propuesta para la mejora del aprendizaje de la programación concurrente**. En este paquete se realizan las tareas pertinentes para mejorar el aprendizaje del paradigma de la programación concurrente y en tiempo real.

- **PT4. Evaluación y caso de estudio para la programación concurrente y en tiempo real**. En este paquete se evalúa la propuesta anterior con alumnos para validar el enfoque planteado, utilizando una serie de cuestionarios con preguntas tipo *test* y una actividad a resolver.

A modo aclarativo, el diagrama de *Gannt* presentado en la Figura 4.12 muestra el tiempo estimado para cada PT y T. En las siguientes subsecciones, se detalla todo lo realizado en cada PT.

4.2.1 PT1. Propuesta de una metáfora de carreteras y señales de tráfico para soportar la visualización dinámica de programas

Este PT está destinado a la modificación de una metáfora definida en un TFG previo. Se pretende que mediante esta adaptación se habilite la generación de visualizaciones dinámicas que permitan observar la traza de ejecución de un programa. El tiempo empleado para tarea fue de 5 semanas. A continuación se presentan las T involucradas en este PT.

- **T1. Representación del concepto de hilo de ejecución**. A lo largo de esta tarea se diseñó y elaboró un modelo basado en un camión para representar el hilo de ejecución de un programa.
- **T2. Representación del concepto de punto de ruptura**. Durante esta tarea se diseñó y elaboró un modelo basado en un cono de carreteras para representar el punto de ruptura en un programa.

4.2.2 PT2. Diseño y codificación de un sistema de visualización dinámica

En este PT se engloban todas las T relacionadas con el soporte computacional para la visualización dinámica de programas. Las T involucradas en este PT abarcan desde el estudio de información y análisis de propuestas, hasta la implementación y validación de prototipos. Por ello, este PT se dilata considerablemente en el tiempo al ser el núcleo principal del proyecto. A continuación, se desglosan las T que forman parte de este PT.

- **T1. Estudio del entorno de depuración de Eclipse**. Esta tarea se ha diseñado para conocer en profundidad el entorno de depuración de Eclipse. El objetivo es conocer las herramientas que este entorno proporciona, cuáles son aquellas que interesan en el contexto de este trabajo y cuál es el procedimiento para acceder a ellas de forma programática. El tiempo para esta tarea ha sido de 2 semanas.
- **T2. Creación de punto de ruptura en tiempo de ejecución**. Durante esta tarea se estudió la forma de añadir un punto de ruptura en el fragmento de código que el usuario quiere visualizar con el dispositivo de RA. Esta tarea es el paso intermedio para poder desplegar el depurador. Dada la dificultad de la tarea al tener que añadir un elemento en tiempo de ejecución, el tiempo empleado ha sido de 2 semanas.

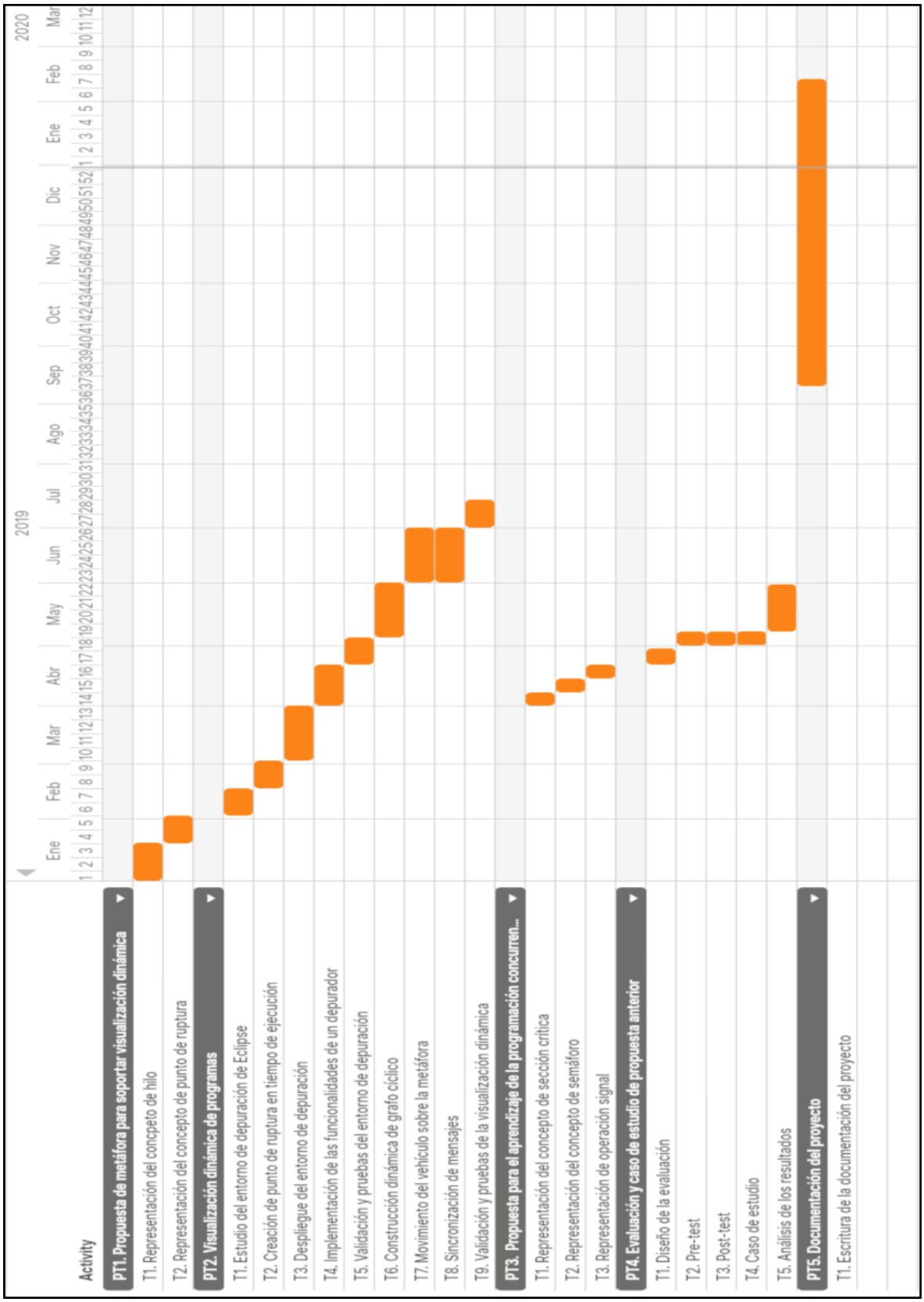


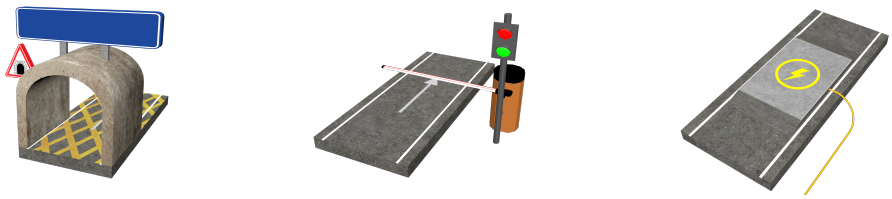
Figura 4.12: Desglose de tareas y tiempo empleado

- **T3. Despliegue del entorno de depuración**. A lo largo de esta tarea se implementó un procedimiento para desplegar de manera óptima en términos de tiempo el depurador de Eclipse. Además, se definió una estructura en formato JSON para almacenar la información relativa a la depuración y mejorar así su gestión. Dada la complejidad de la tarea, el tiempo empleado para su finalización fue de 4 semanas.
- **T4. Implementación de las funcionalidades de un depurador**. Esta tarea sirvió para implementar las funcionalidades de *step over* y *step up to a point* típicas en la mayoría de depuradores actuales. Su realización está marcada por el hecho de que el dispositivo de RA necesita conocer la evaluación de cada sentencia para efectuar la visualización. Dicha tarea fue realizada en 3 semanas.
- **T5. Validación y pruebas del entorno de depuración**. Para la realización de esta tarea se utilizaron un conjunto de algoritmos a fin de identificar fallos en el prototipo que permitiesen obtener una versión estable. Los algoritmos utilizados fueron: *bubble-sort*, *fibonacci*, *linear-search*. El tiempo dedicado a esta tarea fue de 2 semanas.
- **T6. Construcción dinámica de grafo cíclico dirigido**. Esta tarea se centra en la implementación de un procedimiento para la construcción de un grafo cíclico dirigido sobre la metáfora de carreteras y señales de tráfico. La idea es añadir en cada representación gráfica varios puntos de referencia que permitan guiar al vehículo en la visualización, aparte de enviar la información contenida en cada representación al *plug-in* de visualización. Esta estructura son puntos de referencia (o *waypoints* en inglés), las cuales mantienen la información del nodo actual y siguiente. Dada la complejidad de esta tarea, el tiempo empleado para su culminación fue de 4 semanas.
- **T7. Movimiento del vehículo sobre la metáfora**. Durante esta tarea se implementó un procedimiento para orientar y mover el vehículo, teniendo en cuenta el punto de referencia sobre el que está y por el cual debe de pasar. Esta tarea tuvo una duración de 4 semanas.
- **T8. Sincronización de mensajes**. A lo largo de esta tarea se implementó un algoritmo para sincronizar el intercambio de mensajes entre el cliente y servidor. El cliente envía un mensaje al servidor, solicitando información sobre la sentencia que el vehículo tiene prevista ejecutar. Por otro lado, el servidor recibe dicho mensaje, reenviando éste con la información solicitada. El tiempo empleado para la finalización de esta tarea fue de 4 semanas.
- **T9. Validación y pruebas de la visualización dinámica**. Esta tarea sirvió para identificar errores. Por un lado sirvió para identificar si el envío y recepción de archivos JSON se realizaba correctamente. Además, se pudo comprobar si la codificación y/o descodificación de estos archivos se realizaba adecuadamente. Por otro lado, esta tarea también permitió identificar si el movimiento del vehículo era o no correcto como

consecuencia de la generación del grafo que soporta el movimiento de éste. El tiempo empleado para esta tarea fue de 2 semanas.

4.2.3 PT3. Propuesta para la mejora del aprendizaje de la programación concurrente

Este PT tiene el objetivo de establecer una primera aproximación para hacer frente a las dificultades que estudiantes presentan a la hora de trabajar en programas con varios hilos de ejecución. Para ello, se pretende diseñar y elaborar un conjunto de modelos alineados con la metáfora que establezcan una relación de correspondencia entre conceptos y mecanismos de sincronización de este paradigma con elementos del mundo real. La Figura 4.13 presenta los nuevos modelos diseñados para esta propuesta.



(a) Sección crítica (b) Semáforo (c) Operación signal

Figura 4.13: Propuesta de metáfora para el aprendizaje de la programación concurrente

A continuación, se desglosan las T utilizadas para completar este paquete, entre las cuales suman una duración de 2 semanas.

- **T1. Representación del concepto de sección crítica**. En esta tarea se diseñó y elaboró un modelo basado en un tramo de carreteras con franjas diagonales amarillas (ver Figura 4.13a).
- **T2. Representación del concepto de semáforo**. A lo largo de esta tarea se diseñó y construyó un modelo basado en una señal de control de tráfico (ver Figura 4.13b).
- **T3. Representación de operación signal**. Durante esta tarea se diseño y modeló una figura basada en un interruptor sobre un tramo de carretera con el objetivo de notificar a un semáforo que un vehículo ha pasado por éste (ver Figura 4.13c).

4.2.4 PT4. Evaluación y caso de estudio para la programación concurrente

Este PT está definido para evaluar la propuesta de una metáfora para el paradigma de programación citado, mediante el uso de cuestionarios y una actividad a modo de caso de estudio. A continuación, se presentan las T involucradas en este PT, cuya duración fue de 4 semanas.

- **T1. Diseño de la evaluación**. En esta tarea se diseñó la prueba que los estudiantes realizarían durante una clase como prueba para evaluar la metáfora.

- **T2. Pre-test**. Tarea destinada para que los estudiantes contesten a una serie de preguntas relacionadas con sus conocimientos sobre programación.
- **T3. Post-test**. Tarea destinada a que los estudiantes contesten preguntas sobre la adecuación y comprensión de la metáfora, así como la transcripción de una visualización a pseudocódigo o lenguaje de programación.
- **T4. Caso de estudio**. Actividad propuesta a los estudiantes donde se visualiza un algoritmo concurrente mediante la metáfora de carreteras y señales de tráfico.
- **T5. Análisis de los resultados**. En esta tarea se analizaron los resultados obtenidos tras la realización de la prueba.

4.3 Metodología de trabajo

Dada la naturaleza del proyecto, se ha necesitado de una metodología que permitiera trabajar de forma paralela, añadiendo nuevas características y funcionalidades. Además, esto ha permitido la corrección de errores de cada componente, verificando su funcionamiento para integrarlos de forma satisfactoria en el sistema final.

El planteamiento que se ha llevado a cabo para seguir el continuo desarrollo del proyecto ha sido mediante reuniones cada 1 o 2 semanas. El propósito de las reuniones semanales han tenido como fin establecer los objetivos a realizar en la semana posterior y comprobar el estado del trabajo actual.

Cabe destacar que debido a la arquitectura del sistema orientado a componentes y subsistemas, éstos han podido ser desarrollados de forma aislada sin requerir su total integración en el proyecto final. Una vez un componente o subsistema estaba implementado, se realizaban las pruebas pertinentes para dar por completado el hito correspondiente y pasar al siguiente.

4.3.1 Desarrollo iterativo e incremental

El presente proyecto ha seguido una metodología inspirada en el proceso de **desarrollo iterativo e incremental** [Coc08], la cual plantea una serie de iteraciones proyectadas en un periodo corto de tiempo para ser examinadas en cualquier instante.

Esta metodología es principalmente usada en el mundo del desarrollo *software*, la cual permite mantener o adaptar el mismo para que los cambios que se produzcan sean fáciles de integrar en el proyecto. Estas adaptaciones se consiguen gracias a al uso de iteraciones en pequeñas porciones de tiempo, que tratan de mejorar los componentes que ya han sido desarrollados. Una vez se ha comprobado que la funcionalidad es la deseada, se continúa con las iteraciones hasta finalizar los hitos que se establecieron en etapas iniciales.

La Figura 4.14 muestra gráficamente el proceso anteriormente comentado, donde una iteración se define como un desarrollo cíclico compuesto por las fases de análisis de requisitos,

diseño, implementación y pruebas. A continuación, se explica en detalle los puntos clave que caracterizan a esta metodología ágil junto a su aplicación en el desarrollo del proyecto.

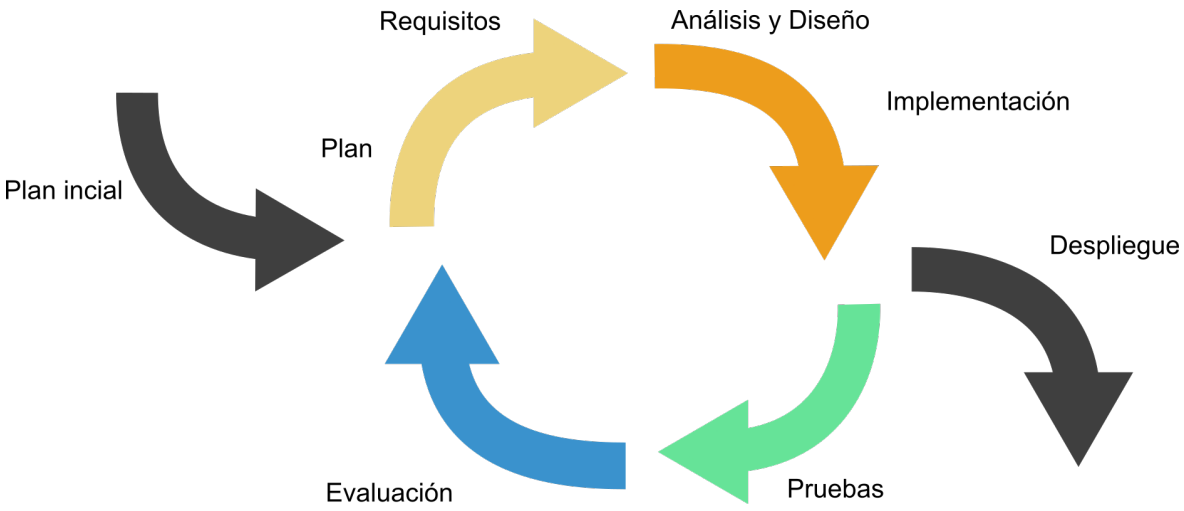


Figura 4.14: Modelo de desarrollo iterativo e incremental

4.3.1.1 Desarrollo incremental

El desarrollo incremental es una estrategia de organización y programación en el cual varias partes del sistema son desarrolladas en diferentes momentos. El trabajo es dividido en tareas, las cuales son programadas en el tiempo con el fin de ser integradas como un todo en el sistema final.

Esta separación de trabajo permite trabajar individualmente sobre una tarea específica, de tal modo que éstas no interfieran entre sí. En este sentido, el presente proyecto ha seguido esta estrategia a la hora de estructurar las funcionalidades del sistema. Esto ha permitido separar las responsabilidades del trabajo con el objetivo de trabajar de manera paralela en cada uno de los módulos o subsistemas.

4.3.1.2 Desarrollo iterativo

El desarrollo iterativo es una estrategia en la cual varias partes del sistema son revisadas o inspeccionadas para mejorar o corregir errores de las iteraciones anteriores. Normalmente, las partes que suelen ser examinadas por el equipo de desarrollo están relacionadas con las secciones técnicas, debido a que el proyecto es susceptible de cambios.

En el caso del sistema expuesto en este documento, esta metodología ha sido utilizada para revisar y modificar todos los componentes desarrollados. Esto ha permitido corregir errores y dotar al sistema de robustez para evitar problemas que pudieran producirse durante la ejecución.

4.3.2 Iteraciones

Este sistema se engloba dentro de un proyecto de gran envergadura, lo que ha derivado en realizar varias tareas prolongadas en el tiempo a fin de ser revisadas una vez completadas. Este proyecto está compuesto por un total de 14 iteraciones. La primera se centra en la recogida de requisitos y la definición del alcance del proyecto. A continuación, las iteraciones se centran en el soporte computacional para visualizar programas de un forma dinámica. Otra parte se centra en la evaluación para validar que el enfoque planteado mejora el aprendizaje de la programación concurrente. El resto se ocupa de un ejemplo en el que se muestra una posible utilización del nuevo planteamiento en un aula.

4.3.2.1 Iteración 1

Durante esta iteración se llevó a cabo la recogida de requisitos para definir los detalles específicos que deben abordarse en la implementación del sistema. Después de su recogida, se definió el alcance del proyecto, a fin de estimar el tiempo y recursos que se emplearían.

Los requisitos que se obtuvieron fueron:

- Se diseñarán una serie de modelos 3D para soportar la visualización dinámica de programas.
- El sistema será capaz de lanzar el depurador de Eclipse cuando el usuario lo indique desde el dispositivo de RA.
- El sistema será capaz de introducir puntos de ruptura cuando el usuario lo indique en la visualización.
- El sistema será capaz de mover el vehículo sobre la metáfora simulando ejecutar sentencias.
- El sistema evaluará de forma dinámica las sentencias por las que pase el vehículo.
- La velocidad del vehículo se calculará en función del tamaño del algoritmo a representar.
- La interacción con la visualización será lo más intuitiva posible para mejorar el proceso de aprendizaje.
- Se diseñarán una serie de modelos 3D para soportar la visualización de programas concurrentes.
- Se diseñará una prueba con la que evaluar el enfoque planteado para la programación concurrente.
- Se diseñará una metodología a modo de ejemplo para mostrar una posible utilización de la propuesta para la programación concurrente en el aula.

El Tabla 4.1 muestra información sobre las fechas en las que se realizó la iteración, el tiempo en horas que se dedicó y las T y PT afectados.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
Todos los PT	Todas las T	Recogida de requisitos. Definición del alcance	20/12/2018	28/12/2018	10h

Tabla 4.1: Descripción resumida de la primera iteración

4.3.2.2 Iteración 2

En esta tarea se llevó a cabo la modificación de una metáfora para soportar visualizaciones dinámicas. Aparte, se realizó un estudio sobre el entorno de depuración de Eclipse para utilizar sus herramientas en iteraciones posteriores. El Tabla 4.2 muestra información adicional sobre esta iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT1	T1, T2	Representación del concepto de hilo. Representación del concepto de punto de ruptura	01/01/2019	01/02/2019	50h
PT2	T1	Estudio de Eclipse	02/02/2019	12/02/2019	28h

Tabla 4.2: Descripción resumida de la segunda iteración

4.3.2.3 Iteración 3

A lo largo de esta tarea se implementó un procedimiento encargado de desplegar el depurador de Eclipse. Complementariamente, se realizaron los primeros pasos para sincronizar el despliegue entre el dispositivo de RA y el *plug-in* de Eclipse. Véase la Tabla 4.3 para conocer más detalles sobre esta iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T3	Despliegue del depurador de Eclipse	10/03/2019	05/05/2019	50h

Tabla 4.3: Descripción resumida de la tercera iteración

4.3.2.4 Iteración 4

En esta iteración se llevó a cabo la implementación relacionada con la creación de puntos de ruptura en tiempo de ejecución. Adicionalmente, se realizaron una serie de pruebas para

validar el desarrollo realizado. Véase la Tabla 4.4 para obtener más información sobre las T involucradas, PT y tiempo dedicado.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T2	Creación de puntos de ruptura de forma programática. Validación de nueva funcionalidad	13/02/2019	08/03/2019	40h

Tabla 4.4: Descripción resumida de la cuarta iteración

4.3.2.5 Iteración 5

En esta iteración se implementó uno de los métodos básicos que integra un depurador, *step over*. Esta implementación es una de las partes más importantes, ya que permite que el cliente conozca el resultado de una sentencia previamente enviada al servidor. Por ello, se realizaron una serie de pruebas para obtener una versión completa con esta funcionalidad. En el Tabla 4.5 se muestra más información sobre esta iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T4, T5	Implementación de la funcionalidad *step over*. Validación de nueva funcionalidad	02/04/2019	14/05/2019	15h

Tabla 4.5: Descripción resumida de la quinta iteración

4.3.2.6 Iteración 6

En esta iteración se implementó otra funcionalidad básica que los depuradores actuales integran, *step up to a point*. Al igual que con la iteración anterior, se realizaron una serie de pruebas a partir de la finalización de este desarrollo, a fin de obtener una versión estable con los nuevos cambios. Véase la Tabla 4.6 en el cual se presenta información adicional.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T4, T5	Implementación de *step up to a point*. Validación de nueva funcionalidad	14/04/2019	26/05/2019	10h

Tabla 4.6: Descripción resumida de la sexta iteración

4.3.2.7 Iteración 7

Durante esta iteración se diseñó una estructura con la que el vehículo pudiera moverse por la representación en función del resultado de cada sentencia. Para ello, esta estructura se basó en un grafo cíclico dirigido, con el que conocer los puntos adyacentes a un nodo y así facilitar el movimiento y dirección del camión. Véase la Tabla 4.7 para conocer más detalles sobre la iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T5, T6	Construcción dinámica de grafo cíclico dirigido. Validación de nueva funcionalidad	14/04/2019	26/05/2019	35h

Tabla 4.7: Descripción resumida de la séptima iteración

4.3.2.8 Iteración 8

A lo largo de esta iteración se realizó el desarrollo relacionado con el movimiento del vehículo. Para ello se utilizó la estructura implementada en la iteración anterior. Véase la Tabla 4.8 para conocer las T involucradas, los PT y el tiempo dedicado.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT2	T5, T6	Construcción dinámica de grafo cíclico dirigido. Validación de nueva funcionalidad	14/04/2019	26/05/2019	35h

Tabla 4.8: Descripción resumida de la octava iteración

4.3.2.9 Iteración 9

En esta iteración se hizo especial hincapié en la sincronización de mensajes entre cliente y servidor, además de la realización de una serie de pruebas para validar el desarrollo. Así, se pudo obtener una versión totalmente funcional con la que probar el sistema con múltiples algoritmos para comprobar su capacidad. Véase la Tabla 4.9 para conocer más detalles sobre las T y tiempo dedicado en esta iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT3	T8, T9	Sincronización de mensajes. Validación de la nueva funcionalidad	15/06/2019	17/07/2019	50h

Tabla 4.9: Descripción resumida de la novena iteración

4.3.2.10 Iteración 10

Esta iteración se centra en el diseño de nuevos modelos 3D enfocados en mejorar la comprensión de conceptos relacionados con la programación concurrente. En el Tabla 4.10 se muestra más información sobre esta iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT3	T1	Diseño de sección crítica	02/04/2019	10/04/2019	3h
PT3	T2	Diseño de semáforo	11/04/2019	18/04/2019	3h
PT3	T3	Diseño de operación signal	20/04/2019	28/04/2019	3h

Tabla 4.10: Descripción resumida de la décima iteración

4.3.2.11 Iteración 11

Durante esta iteración se diseñó y preparó el experimento para evaluar la metáfora propuesta, es decir, valorar si los diseños creados en la iteración anterior mejoran la comprensión de conceptos y mecanismos de sincronización relacionados con la programación concurrente. En el Tabla 4.11 se muestra información complementaria.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT4	T1	Diseño de la prueba piloto	19/04/2019	28/05/2019	10h

Tabla 4.11: Descripción resumida de la undécima iteración

4.3.2.12 Iteración 12

A lo largo de esta iteración se realizó la evaluación con estudiantes de la asignatura de Sistemas Operativos II de la Escuela Superior de Informática. En esta evaluación se valoraron los conocimientos previos de los estudiantes y cómo la nueva metáfora influye en aquellos estudiantes que más problemas tienen en afrontar problemas en los que es necesario trabajar con varios hilos de ejecución. Véase la Tabla 4.12 para más información.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT4	T2, T3, T4	Evaluación de la prueba con estudiantes	01/05/2019	01/05/2019	2h

Tabla 4.12: Descripción resumida de la duodécima iteración

4.3.2.13 Iteración 13

Esta iteración se centra en el análisis de los datos recogidos tras la finalización de la evaluación de la metáfora planteada. Así, se pudo comprobar estadísticamente si realmente la metáfora ayuda en el proceso de aprendizaje. En el Tabla 4.13 se recoge información adicional a la iteración.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT4	T5	Análisis de los datos recogidas de la evaluación	02/04/2019	15/05/2019	15h

Tabla 4.13: Descripción resumida de la decimotercera iteración

4.3.2.14 Iteración 14

Esta iteración se centra en la elaboración del presente documento. En el Tabla 4.14 se obtiene información sobre el tiempo involucrado para esta T.

Paquete de trabajo	Tareas afectadas	Objetivos alcanzados	Fecha inicio	Fecha fin	Tiempo estimado
PT5	T1	Escritura de la documentación del proyecto	10/09/2019	12/02/2020	110h

Tabla 4.14: Descripción resumida de la decimocuarta iteración

4.4 Medios empleados para el desarrollo

En esta sección se exponen todos los medios que se han utilizado para llevar a cabo el presente proyecto. Por un lado, se especifican los medios *hardware* que se han previsto necesarios para el despliegue del sistema. Por otro lado, los medios *software* (lenguaje, entorno de desarrollo, herramientas de gestión y planificación, etc.) requeridos para su construcción, así como las bibliotecas, *frameworks*, *plug-ins*, etc., que han tenido mayor impacto en su desarrollo.

4.4.1 Medios *hardware*

Esta sección se exponen y detallan los medios *hardware* utilizados para la realización de este sistema.

- **Microsoft HoloLens**. Este dispositivo pertenece al grupo de investigación CHICO de la ESI de la UCLM. Dicho dispositivo se ha utilizado para sumergir al usuario en un entorno 3D virtual. Su elección se debe a la posición dominante en el mercado por las amplias capacidades que ofrece para mostrar e interactuar con la visualización, así como otras facilidades para reconstruir virtualmente el entorno real donde se encuentra el usuario, utilizado para facilitar el posicionamiento de la visualización en el espacio.
- **Computador**. Para el desarrollo del sistema se ha utilizado un equipo que cumpla con los requisitos que Microsoft especifica en su web, a fin de crear aplicaciones para el dispositivo de RA[8]. Las características con las que cuenta el computador son:
  - Windows 10 Education de 64-bit.
  - CPU con 4 núcleos.
  - GTX 1050 Ti.
  - 16 GB de memoria RAM.

4.4.2 Medios *software*

Esta sección explica de manera sintetizada los medios *software* utilizados en este proyecto, los cuales se plasman en forma de esquema en la Figura 4.15.

4.4.2.1 Sistema operativos

- **Windows 10 Fall Creators Update**. Para la elaboración del trabajo se ha elegido este sistema operativo debido a la plataforma de Realidad Mixta de Windows (término en inglés Windows Mixed Reality (WMR)), la cual permite desarrollar aplicaciones para el dispositivo de RA de Microsoft.

[8] https://docs.microsoft.com/es-es/windows/mixed-reality/install-the-tools

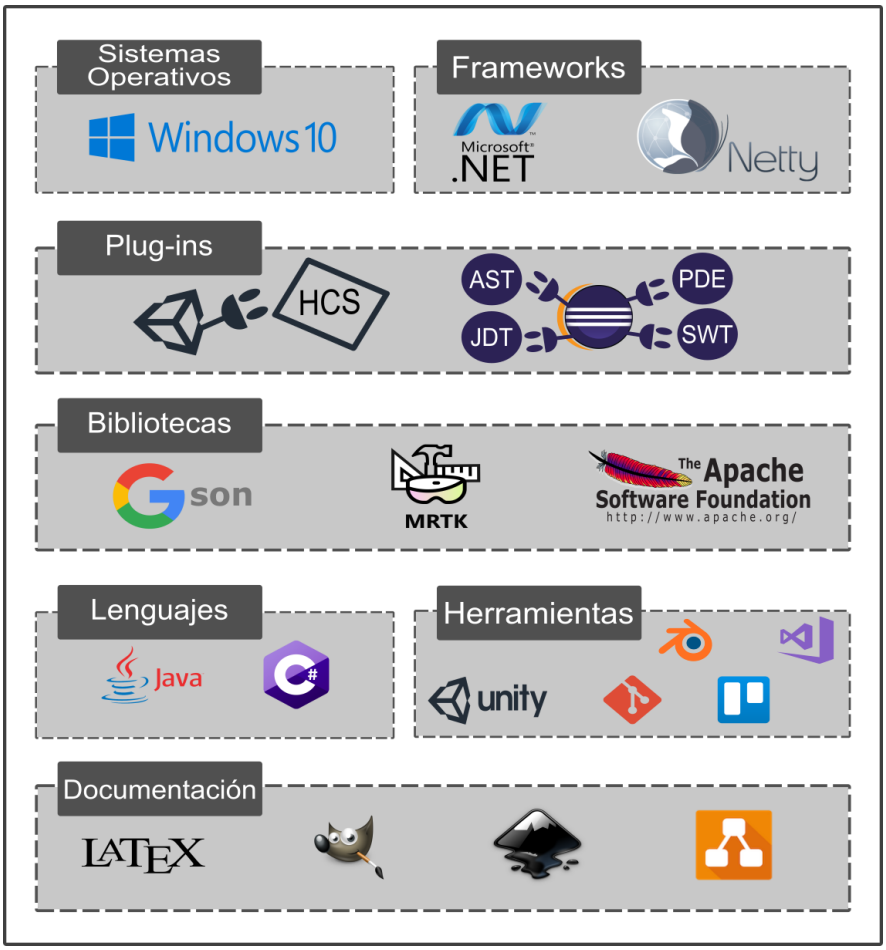


Figura 4.15: Medios *software*

4.4.2.2 *Frameworks*

- **.NET 4.6**. Es un *framework* de Microsoft que proporciona un conjunto de servicios a las aplicaciones en ejecución. Consta de dos componentes principales: Common Language Runtime (CLR), quiera es el motor de ejecución que controla las aplicaciones en ejecución, y la biblioteca de clases del *framework* de .NET, las cuales están dedicadas al tratamiento de datos, herencias y patrones de diseño, entre otros. Algunos de los diferentes servicios que ofrece .NET a las aplicaciones en ejecución son:
  - Administración de memoria de forma transparente para el desarrollador.
  - Sistema de tipos comunes que implica una mayor interoperabilidad entre aquellos lenguajes compatibles con .NET *framework*.
  - Biblioteca de clase para controlar operaciones de bajo nivel.

- **Netty**[9]. Es un *framework* de aplicación asíncrono dirigido por eventos para el desarrollo rápido de aplicaciones en Java basadas en la arquitectura cliente-servidor. Esto es utilizado para simplificar la programación de servidores que utilizan *socket* TCP y UDP. Para el desarrollo del trabajo se ha empleado la versión 4.1.16.
- **Java Abstract Syntax Tree (AST)**. El Árbol de Sintaxis Abstracta de Java es un *framework* que es utilizado por muchas de las herramientas del IDE de Eclipse, en las cuales se incluyen la refactorización, corrección y asistencia rápida. Este *framework* convierte el código fuente de un archivo Java en una estructura arbórea, con el objetivo de facilitar su análisis y modificación de forma programática.
- **JDT Debug**. Modelo compuesto de varios *plug-ins* para el soporte de la ejecución y depuración de códigos en lenguaje Java.

4.4.2.3 *Plug-ins*

- **HoloLensCameraStream**[10]. Es un *plug-in* de Unity 3D para recuperar en tiempo real los fotogramas de la cámara de las gafas de RA de Microsoft. Esto permite utilizar la cámara del dispositivo para realizar tareas de visión por computador y aprendizaje automático, entre otras. Para el desarrollo del trabajo se ha utilizado la versión 0.3.0.
- **Eclipse Plug-in Development Environment (PDE)**. El Entorno de Desarrollo de *Plug-ins* de Eclipse es un entorno para crear, desarrollar, probar, desplegar y construir *plug-ins* para dicho entorno. Éste además proporciona componentes del *framework* de Java Open Services Gateway initiative (OSGI), el cual lo convierte en un entorno ideal para la programación de componentes.

  PDE se divide en tres componentes principales:

  - IU. Proporciona editores, asistentes, vistas y otras herramientas para la creación de un entorno con características completas para el desarrollo de *plug-ins* y paquetes OSGI.
  - Application Programming Interface (API). Herramienta que permite a los desarrolladores identificar incompatibilidades binaria entre dos versiones, números de versión de *plug-ins* incorrectos, proporcionar documentación del código, etc.
  - Construcción. Facilita la automatización del proceso de creación de *plug-ins* gracias a los scripts producidos en tiempo de desarrollo.

- **Standard Widget Toolkit (SWT) designer**. Es un *plug-ins* desarrollado por Eclipse para la creación de interfaces gráficas. Éste hace uso de un editor visual en el cual se pueden crear interfaces sin la necesidad de escribir código Java, ya que éste es generado automáticamente a partir de la descripción visual proporcionada.

[9] `https://github.com/netty/netty`

[10] `https://github.com/VulcanTechnologies/HoloLensCameraStream`

4.4.2.4 Bibliotecas

- **MediaFrameQrProcessing**[11]. Es una biblioteca de código abierto destinado al reconocimiento de códigos QR basada en otra biblioteca llamada ZXing.Net[12].
- **MixedRealityToolkit-Unity**[13]. Es una colección de scripts y componentes con la intención de acelerar el desarrollo de aplicaciones dirigidas a los dispositivos de RA existentes en la actualidad haciendo uso de la versión 2017.2.1.4. MixedRealityToolkit-Unity usa código basado en MixedRealityToolkit (MRTK), el cual proporciona una biblioteca de clases extensa para el reconocimiento de superficies, interacción con hologramas y simulación de sonidos en un entorno 3D, entre otros.
- **QRGen**[14]. Sencilla API para la generación de códigos QR en Java construida sobre ZXing[15]. Se ha hecho uso de la versión 2.3.0.
- **Gson**[16]. Biblioteca de Java usada para convertir objetos de este lenguaje en su representación JSON. Para el desarrollo del proyecto se ha empleado la versión 2.8.2.
- **Apache Commons Lang**[17]. Es una biblioteca que proporciona una serie de ayudas para el desarrollo de aplicaciones en Java, particularmente en la manipulación de cadenas, concurrencia, reflexión y operaciones matemáticas de negocios, entre otras muchas contribuciones. Se ha empleado la versión 3.5.

4.4.2.5 Lenguajes de programación

- **Java**. Lenguaje de programación creado por Sun Microsystems el cual se caracteriza por ser de propósito general, concurrente y POO. La elección de Java se debe a que es el único lenguaje utilizado para la creación de *plug-ins* para la plataforma de desarrollo de Eclipse. Esto se debe al compilador interno que la herramienta incorpora junto a un módulo completo de los archivos fuente de Java.
- **C#**. Lenguaje de POO desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. La utilización de este lenguaje se debe al soporte que tiene el entorno Unity 3D para la generación de scripts.

[11] https://github.com/mtaulty/QrCodes
[12] https://www.nuget.org/packages/ZXing.Net
[13] https://github.com/Microsoft/MixedRealityToolkit-Unity
[14] https://github.com/kenglxn/QRGen
[15] https://github.com/zxing/zxing
[16] https://github.com/google/gson
[17] https://github.com/apache/commons-lang

4.4.2.6 Herramientas de desarrollo

- **Visual Studio 2017**. Para la edición de código se ha utilizado dicho editor, ya que permite aprovechar las herramientas de depuración y *profiling* dedicadas a C#. Además, este editor permite desplegar las soluciones tanto en el dispositivo de RA como en el emulador, el cual es proporcionado por Microsoft para simular la visualización que realiza el dispositivo.
- **Unity 3D**. Es un motor de juegos que permite crear aplicaciones o videojuegos para las gafas de RA de Microsoft, entre un conjunto extenso de dispositivos. Este entorno ha permitido representar los modelos 3D, de tal modo que HoloLens lo visualice conforme se ha definido en el motor de juegos. Se ha hecho uso de la versión 2017.4.9f1.
- **Blender**[18]. Entorno multiplataforma el cual está dedicado a la generación de modelos, animaciones, iluminación y renderizado, entre otros. Su utilización se debe a que Unity 3D nativamente importa archivos `.blend`, lo que facilita la utilización de modelos creados en el motor de juegos. Cabe destacar que todos los modelos 3D integrados en el sistema han sido realizados con esta herramienta. Para el desarrollo del proyecto se ha usado al versión 2.8.
- **Eclipse versión 2018-12-R**. Es un plataforma de desarrollo diseñada para la creación de aplicaciones, la cual ha permitido la creación de *plug-ins* mediante PDE. Su uso se debe a la familiarización que presentan los alumnos con esta herramienta.
- **Hercules**[19]. Herramienta desarrollada para trabajar como un terminal el cual puede manejar puertos serie y protocolos UDP/IP y TCP/IP. La utilización de esta herramienta se debe a la facilidad de simular y probar comunicaciones TCP/IP entre cliente y servidor. Se ha hecho uso de la versión 3.2.8.
- **Git**. Sistema de control de versiones que ha permitido almacenar un repositorio con el código y documentación del proyecto. El servicio de repositorio utilizado ha sido proporcionado por BitBucket debido a que los proyectos pueden ser creados de manera privada, además de proporcionar un límite de hasta 2 GB por repositorio de manera gratuita. Se ha hecho uso de la versión 4.2.1.
- **Trello**[20]. Sistema web para la administración de proyectos mediante el uso de tarjetas virtuales organizadas sobre un tablero virtual. Las tarjetas han representado los hitos correspondientes a los objetivos destacados en el capítulo de objetivos (ver Capítulo 2).

[18] `https://www.blender.org/`
[19] `https://www.hw-group.com/software/hercules-setup-utility`
[20] `https://trello.com/`

4.4.2.7 Documentación

- **LaTeX**. Sistema de elaboración de textos utilizando la distribución TeX Live en su versión lanzada en 2018. El editor usado para la escritura del documento ha sido Visual Studio Code, debido al resaltado sintáctico que proporciona y la corrección ortográfica interactiva.
- **Gimp**[21]. Programa tanto de edición de imágenes digitales como de dibujo para la modificación de figuras insertadas en el documento. Se ha empleado la versión 2.10.2.
- **Inkscape**[22]. Es un editor que trabaja con gráficos vectoriales libre y de código abierto utilizado para crear y editar diagramas, gráficos, logotipos, etc. Para este trabajo se ha empleado la versión 0.92.
- **Draw.io**[23]. Aplicación web usada para la realización de los diagramas insertados en el documento.

[21] https://www.gimp.org/
[22] https://inkscape.org/es/
[23] https://www.draw.io/

Capítulo 5

Resultados

Una vez descrito el problema, los objetivos, la metodología empleada y la arquitectura y las decisiones de diseño adoptadas, se exponen los resultados en términos del producto final al que se ha dado salida (demostración de que el desarrollo realizado está asociado con lo descrito en este trabajo) y de la consecución de los objetivos previamente planteados. Adicionalmente, se expone el desglose de costes que ha supuesto el desarrollo de este trabajo y las competencias adquiridas. Finalmente, se enumeran las publicaciones científicas, así como los premios que han surgido como parte de la idea que subyace de este proyecto.

5.1 Visualización dinámica de programas

Esta sección trata de demostrar la parte dinámica de la ejecución de un programa. A modo de ejemplo y para situar la descripción, se considera el algoritmo de la Búsqueda Lineal de un número (ver Listado 5.1), utilizado para contar las ocurrencias de un número que aparece consecutivamente en una lista. Además, se ha hecho uso de mosaicos a modo de secuencia de imágenes que han servido de apoyo para acompañar la descripción de la ejecución desde el punto de vista del observador.

```
1  public static int countSerialNums(int n, int [] nums) {
2    int count = 0, res = 0, i = 0;
3    while(i < nums.length) {
4      if(nums[i] == n) {
5        count++;
6      }else {
7        res = max(res, count);
8        count = 0;
9      }
10     i++;
11   }
12   return res;
13 }
```

Listado 5.1: Algoritmo que cuenta las ocurrencias de un número que aparece de manera consecutiva en una lista

5.1.1 Preparación de la visualización

En primera instancia, el usuario se dirige al entorno de Eclipse, donde codifica el algoritmo que posteriormente se visualizará en el dispositivo *Microsoft HoloLens*. Para ello, previamente el usuario debe activar el *plug-in* de visualización, que mostrará una vista con tres pestañas, de las cuales una muestra un código QR para que cliente y servidor se conecten.

Una vez codificado el algoritmo, el usuario tiene la posibilidad de visualizarlo mediante una metáfora de carreteras y señales de tráfico. Esto es posible clicando sobre un botón ubicado en la pestaña del código QR. Esto convierte el código fuente a una representación basada en la metáfora mencionada mediante el dispositivo de RA.

En este instante, la representación muestra la estructura estática de un programa, es decir, sentencias y estructuras de control (ver Figura 5.1(1)). El usuario tiene la posibilidad de cambiar este modo de visualización por otro que permite ver dinámicamente la ejecución del programa. Esto se realiza a través de un menú asociado a la visualización. Automáticamente, el depurador de Eclipse es lanzado, reduciendo la interacción del usuario con el sistema y tratando de que se centren fundamentalmente en la visualización.

Entre las acciones disponibles que el usuario puede realizar durante la visualización dinámica, se considera la posibilidad de ejecutar la instrucción sobre la que se encuentra el vehículo, añadir o eliminar puntos de interrupción o incluso clicar sobre las señales de tráfico para observar el resultado de una evaluación.

Con vistas a no realizar una interacción excesiva para visualizar el algoritmo propuesto, el usuario ubica un cono de tráfico (ver Figura 5.1(2)) que representa un punto de interrupción. Así, el vehículo ejecuta sentencias automáticamente hasta llegar hasta este punto, donde el usuario manualmente ejecutará paso a paso cada sentencia.

A modo de ejemplo, la evaluación elegida para describir el algoritmo es la siguiente: (i) *n* se ha inicializado a un valor de 4 para conocer sus ocurrencias dentro de la lista *nums*; (ii) dicha lista se ha definido con los valores [4, 4, 4, 2]. Por lo tanto, el resultado esperado *res* debe ser igual a 3, dado que el número 4 aparece 3 veces de forma consecutiva. Resulta interesante destacar que la elección de la evaluación se realiza en el método *main*, el cual es utilizado para ejecutar cualquier fragmento de código. A continuación, se detalla la evaluación que genera el sistema.

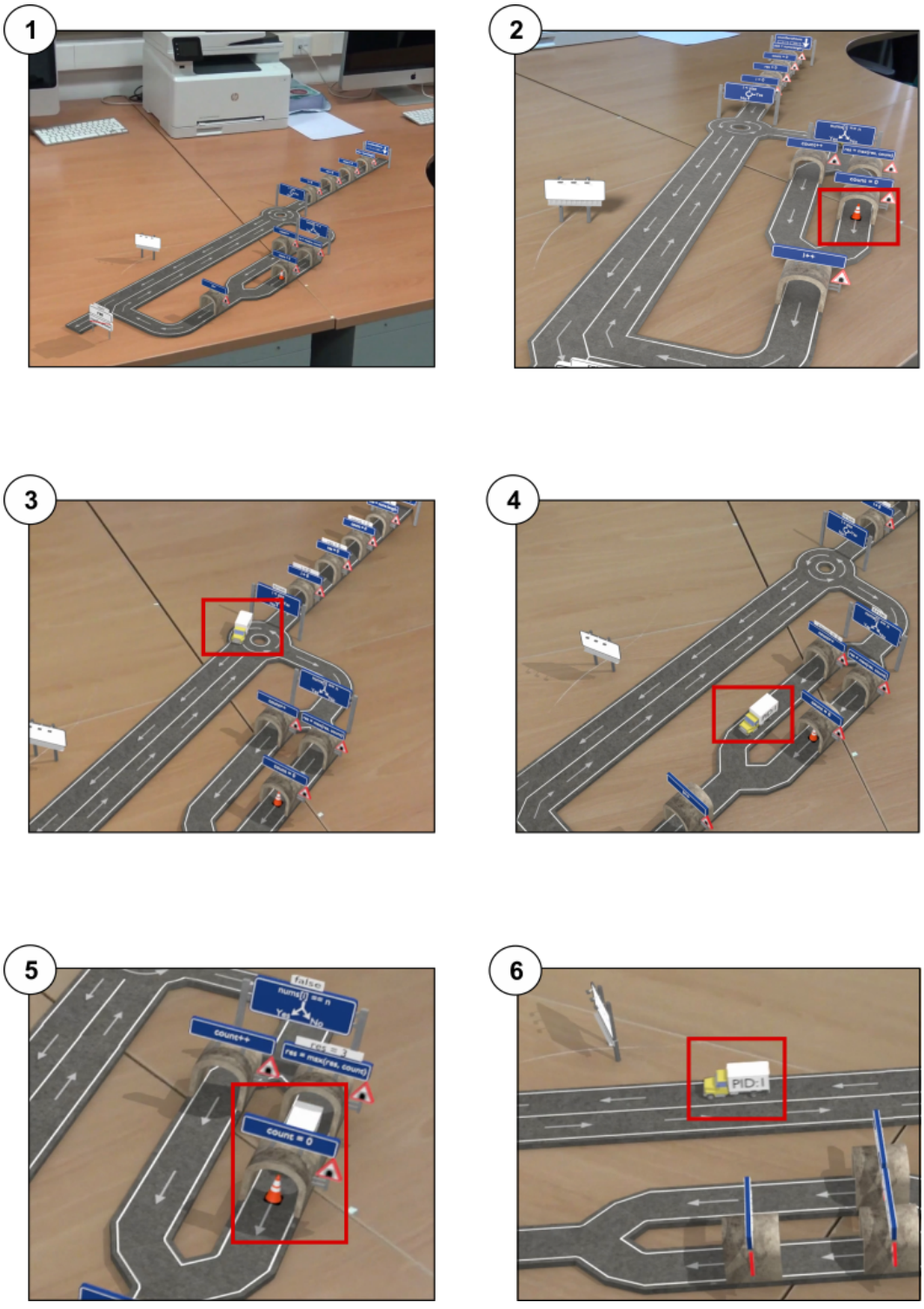


Figura 5.1: Distribución de imágenes que muestran la ejecución del algoritmo

5.1.2 Visualización dinámica del algoritmo Búsqueda Lineal

Se ha comentado anteriormente que el vehículo ejecuta sentencias automáticamente hasta llegar al cono de tráfico que representa un punto de interrupción. Por lo tanto, una vez el vehículo está establecido sobre la representación comienza a circular.

La Figura 5.1(3) muestra el avance del vehículo hasta llegar al bucle *while*. Una vez el vehículo pasa a través de una representación con señal de tráfico, una panel es desplegado para aportar información acerca del resultado de evaluar esa sentencia.

En la primera iteración del bucle, cuando *i = 0*, se aprecia en la Figura 5.1(4) cómo el vehículo pasa por la rama *true*, ya que *nums[i] == n* se evalúa a verdadero, es decir, al ser 4 igual a 4.

En la segunda y tercera iteración, siendo la variable *i* igual a 1 y 2, respectivamente, la trayectoria del vehículo sigue siendo la misma que la anterior, dado que *nums[i]* sigue siendo igual a 4, número elegido al comienzo de la ejecución.

En la cuarta iteración, siendo *i* igual a 3, el vehículo cambia de trayectoria y pasa por la rama *false*, ya que *nums[i]* pasa a ser 2, el último elemento de la lista. En esta rama se obtiene el resultado que retorna el algoritmo, *res = 3*. Aparte, se aprecia que el vehículo se detiene en el cono de tráfico, esperando a que el usuario mediante el gesto que simula un clic de ratón ejecute la funcionalidad *step over* (ver Figura 5.1(5)).

Finalmente, en la Figura 5.1(6), se muestra cómo el vehículo sale del bucle y se dirige hacia el final de la representación, a fin de retornar el resultado generado de ejecutar el algoritmo.

5.2 Encuesta para la evaluación de la metáfora propuesta

Una vez acabado el diseño de los modelos para soportar la visualización de programas concurrentes, el siguiente paso fue realizar su evaluación para poder demostrar que la metáfora ayuda realmente a comprender conceptos y mecanismos asociados al paradigma de la programación concurrente y en tiempo real. Resulta interesante mencionar que, aunque la evaluación se basa en los modelos diseñados para la programación concurrente, también se evalúan los diseños de definición de una función, sentencia condicional, sentencia de bucle, evaluación de expresión, retorno de función, punto de interrupción e hilo de ejecución.

En esta sección se describe la experiencia piloto realizada, basada en un diseño cuasi-experimental [RC08] del tipo pretest - postest, con el cual se midieron aspectos relacionados con el perfil del estudiante, y la comprensión, adecuación y efectividad de la propuesta.

5.2.1 Participantes

En la experiencia participaron un total de 15 alumnos del Grado en Ingeniería Informática de la UCLM, los cuales todo pertenecían a la asignatura de Sistemas Operativos II (SSOOII), intensificación de la rama de Ingeniería de Computadores.

La elección de este grupo se realizó con el fin de contrastar la opinión de alumnos con experiencia previa en la programación de programas multi-hilo. Así, se esperaba obtener una retroalimentación si ellos asocian elementos de la metáfora con conceptos y mecanismos de sincronización que previamente ya han estudiado.

Cada estudiante participó voluntariamente y fue informado que sus datos serían tratados con confidencialidad y utilizados únicamente para dicho estudio. Todos los que realizaron el experimento obtuvieron 0.5 puntos adicionales correspondientes a la valoración de participación con aprovechamiento en clase.

5.2.2 Equipamiento

La experiencia se realizó en las aulas de la ESI de Ciudad Real, concretamente en el aula donde se imparte la asignatura de Sistemas Operativos II (SSOOII). Cada estudiante utilizó su equipo propio, ya que todos ellos llevan con frecuencia su pc a clase. A cada participante se le asignó un sitio concreto, a fin de estar alejados de sus compañeros y/o compañeras de grupo.

5.2.3 Proceso

El propósito de la prueba era analizar la representación gráfica, basada en la metáfora de carreteras y señales de tráfico, para evaluar el nivel de comprensión de estos elementos gráficos, y conocer la opinión de forma subjetiva respecto a la facilidad de entender y la idoneidad de dichos elementos. Esto siempre desde el punto de vista de estudiantes de grado en Ingeniería Informática.

Previo al desarrollo de la experiencia, se utilizaron 10 minutos para poner en contexto a los estudiantes del propósito del proyecto y la funcionalidad del sistema desarrollado hasta la fecha, es decir, la visualización estática y la visualización dinámica de programas con un solo hilo de ejecución. Tras esto, se siguieron los pasos de [WRH+12], formulando el objetivo sobre una plantilla *Goal-Question-Metric*.

El primer paso que tuvieron que realizar los participantes fue rellenar un cuestionario (ver C.1) distribuido en dos secciones con el que se pudiera obtener información relativa al perfil de los estudiantes. El tiempo estimado fue de unos 15 minutos de duración realizado a través de *Microsoft Forms*.

- Información demográfica. Genero, edad, si repiten la asignatura y tipo de acceso al grado.

- La actitud que tienen los estudiantes universitarios hacia la programación (*Computer Programming Attitude Scale for University Students (*CPAS*)*) [CO15]: con esta escala se puede medir la valoración de la utilidad, predisposición y simpatía hacia la programación. Estos aspectos fueron medidos con 19 enunciados (ver C.1, CPAS) usando una escala Likert (1: totalmente en desacuerdo a 5: totalmente de acuerdo). A continuación se desglosan los diferentes CPAS clasificados según su propósito:
  - Simpatía hacia la programación. CPAS1, CPAS3, CPAS5, CPAS9, CPAS11, CPAS14.
  - Utilidad de la programación. CPAS2, CPAS6, CPAS8, CPAS12, CPAS15.
  - Actitud a la programación. CPAS4, CPAS7, CPAS10, CPAS13.
- Otro apartado que tuvieron que rellenar los estudiantes fue sobre sus conocimientos respecto a la programación (*Computer Programming Knowledge Language (*CPKL*)*). ver C.1, CPKL.
  - CPKL. Agrupa conocimientos sobre programación.

Tras completar este cuestionario, los participantes comenzaron la segunda fase, la cual estaba subdividida en tres subfases. El tiempo estimado fue de unos 20 minutos de duración realizado a través de *Microsoft Forms*.

- *Actividad*. En esta tarea (ver C.2, Actividad), se les proporcionó a los estudiantes una visualización en formato de vídeo, la cual debía se transcrita a pseudocódigo, Java o cualquier otro lenguaje de programación que ellos conociesen. La visualización representaba un programa multi-hilo que implementaba alguna solución a algún problema previamente ya estudiado por los estudiantes, adaptado al nivel de conocimiento del grupo de evaluación. Además, se proporcionaron una serie de preguntas enumeradas con posibles soluciones acerca de la ejecución del programa, con el objetivo de comprobar si se había comprendido adecuadamente. Véase este enlace[1] para visualizar el vídeo.

  Previamente a la realización de la tarea, se les proporcionó a los estudiantes una leyenda con el significado de cada representación gráfica, indicando el concepto de programación asociado. Además, se les pidió que puntuasen la facilidad de comprensión de la notación y su idoneidad, a través de una escala de Likert de 5 niveles (1: difícil de entender, 5: fácil de entender), dependiendo de cómo de apropiada y fácil de entender había sido la metáfora. Con fácil de entender, este cuestionario se refiere a si la representación gráfica puede ser comprendida sin necesidad de cualquier tipo de leyenda o entrenamiento previo, mientras que con apropiado se quiere decir si la notación realmente representa el concepto del que se quiere abstraer.
- Tras completar la primera fase de la segunda prueba, los participantes rellenaron un conjunto de preguntas distribuidas en varias secciones. En esta parte de la prueba, los

[1] `https://www.youtube.com/watch?v=66zaRSIjbPA`

participantes tuvieron que responder nuevamente a través de una escala Likert (de 1 a 5) sobre un conjunto de enunciados que sirvieron para medir su *motivación intrínseca y rendimiento subjetivo*, la *carga cognitiva* impuesta por la tarea, la *actitud subjetiva percibida*, y la *calidad de la notación*, basada en la percepción de los estudiantes.

- Motivación intrínseca y rendimiento subjetivo. Esta sección está compuesta por una serie de preguntas (ver C.2, MI) diseñadas para medir la actitud subjetiva percibida (Perceived Subjective Attitude (PSA)), el interés (*Interest (*INT*)*), competencias (*Competencies (*COM*)*), esfuerzo (*Effort (*EFF*)*), y precisión (*Precision (*PRE*)*) del alumno durante la comprensión de la tarea.
- Carga cognitiva. Esta sección incluye una serie de enunciados (ver C.1, CLT) diseñados para medir dos tipos de carga cognitiva propuestos por la Teoría de Carga Cognitiva (*Cognitive Load Theory (*CLT*)*) [PMB10]: (1) carga intrínseca, es decir, la carga asociada a la actividad como tarea demandada (TD), y otros dos enunciados para medir la (2) carga extraña, es decir, la carga derivada de las exigencias (E) de la notación.
- Los participantes también tuvieron que responder su nivel de acuerdo o desacuerdo sobre 14 enunciados (ver C.2) diseñados para medir la percepción de facilidad de uso (*Perception Ease Of Use (*PEOU*)*), percepción de utilidad (*Perception Of Utility (*PU*)*) e intención de uso (*Intention Of Use (*ITU*)*) de la notación. En este estudio, se consideró las dimensiones de calidad para métodos de modelado de requisitos (basados en la percepción de los usuarios) propuestas en [AICG11] e inspiradas en el *framework Technology Acceptance Model (*TAM*)* [Dav89]. Esta propuesta permite predecir la probabilidad de que un método en particular sea aceptado en la práctica, en función de la calidad de los artefactos producidos y la percepción de los usuarios sobre su calidad. Con este propósito, se diseñó un instrumento propio para la evaluación de métodos de modelado de requisitos, realizando ciertos ajustes y modificaciones.

Finalmente, se les dio la oportunidad a los estudiantes de proporcionar algunos comentarios, con el objetivo de hacer futuras modificaciones de la metáfora propuesta si fuese requerido.

5.2.4 Análisis de los resultados

En primera instancia, el primer análisis fue realizado respecto al perfil de los participantes, atendiendo a las variables CPAS (simpatía, utilidad y actitud hacia la programación) y a sus conocimiento sobre programación (CPKL).

La Tabla 5.1 muestra los resultados asociados a la variable CPAS. Aunque ofrece valores tanto de la media como de la desviación típica, centramos la atención en la mediana, al estar los valores basados en una escala de Likert (1 totalmente desacuerdo a 5 totalmente de

acuerdo). Se aprecia, en general, que los estudiantes no muestran demasiada simpatía hacia la programación. Sin embargo, si que están prácticamente todos de acuerdo en que es útil, además de mostrar una alta predisposición. El hecho de que el resultado de la simpatía hacia la programación sea bajo (media=2,61) puede deberse a la carga cognitiva que supone dicha tarea y a la abstracción que requiere.

CPAS	Media*	Mediana
Simpatía hacia la programación	2,61 (.86)	2,00
Utilidad de la programación	3,53 (.97)	4,00
Predisposición a la programación	3,68 (.89)	4,00

Tabla 5.1: Resultado de las variables que están asociadas a la actitud del estudiante con respecto a la programación. *Se muestra la media y la desviación típica (entre paréntesis).

Respecto a los conocimientos en programación, la Tabla 5.2 revela que los estudiantes valoraron de forma positiva sus conocimientos generales en programación, y que tienen experiencia con el lenguaje C y el paradigma basado en programación concurrente.

CPKL	Media*	Mediana
Programación	3,47 (.64)	4,00
Programación en C	3,27 (.59)	3,00
Programación Concurrente	3,13 (.83)	3,00

Tabla 5.2: Resultado de las variables que están asociadas a los conocimientos de programación. *Se muestra la media y la desviación típica (entre paréntesis).

Seguidamente fueron analizados los resultados relacionados con la comprensión e idoneidad de la notación propuesta (ver Tabla 5.3), cuya evaluación fue valorada de forma subjetiva.

Elemento	Comprensión*	Idoneidad*
Definición de función	4,00 (1,00); 4,00	3,93 (1,16); 4,00
Sentencia de bucle	4,47 (0.74); 5,00	4,53 (0.83); 5,00
Sentencia de condición	4,80 (0.56); 5,00	4,73 (0.80); 5,00
Evaluación de expresión	3,67 (0.72); 4,00	3,47 (0.99); 4,00
Retorno de función	4,07 (1,22); 4,00	4,20 (1,08); 4,00
Hilo o proceso	4,47 (0.64); 5,00	4,47 (0.74); 5,00
Sección crítica	3,87 (1,06); 4,00	3,93 (1,03); 4,00
Semáforo	4,73 (0.59); 5,00	4,80 (0.56) 5,00
Operación signal	4,40 (0.83); 5,00	4,20 (1,15); 5,00

Tabla 5.3: Resultado de la comprensión e idoneidad de la metáfora de carreteras y señales de tráfico. *Se muestra la media y la desviación típica (entre paréntesis) seguido de la mediana tras el punto y coma.

En general, la notación tuvo una buena acogida por parte de los estudiantes, siendo 1 el valor más bajo que un elemento de la notación podía obtener, y 5 el valor más alto. Si nos fijamos en la media tanto de la columna *Comprensión* como de la columna *Idoneidad*, prácticamente todos los valores están sobre 4 puntos. Sin embargo, hay dos elementos que no fueron bien valorados: evaluación de expresión y sección crítica. Según los comentarios de algunos alumnos, ambas representaciones no son autoexplicativas y llegan a ser un tanto ambiguas, por ejemplo, en el caso de la evaluación de expresión. Se proporciona además el porcentaje de votos que cada uno de los elementos de la metáfora obtuvo, viendo de forma más clara lo anteriormente mencionado (ver las Figuras 5.2 y 5.3).

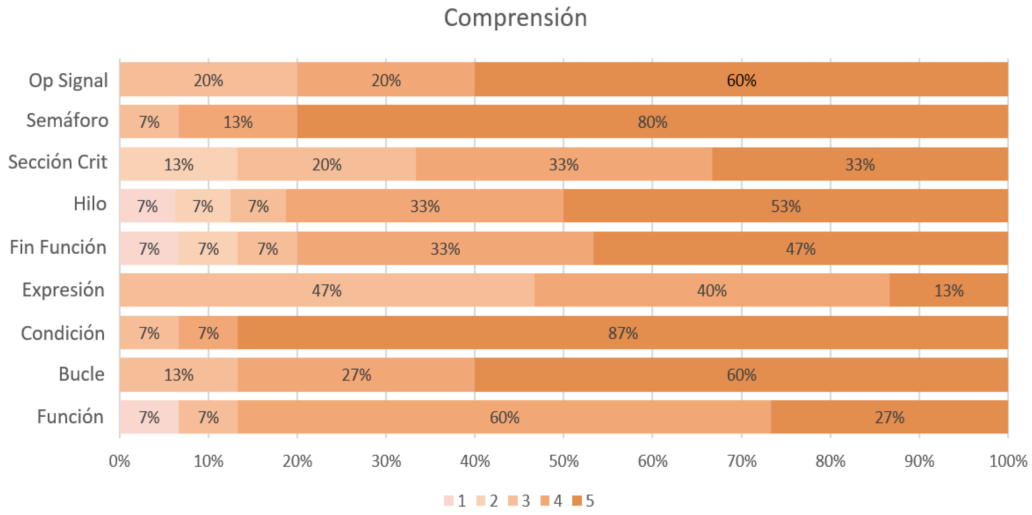


Figura 5.2: Puntuación de comprensión para cada representación gráfica de la metáfora. La escala de colores representa: 1, difícil de entender a 5, fácil de entender.

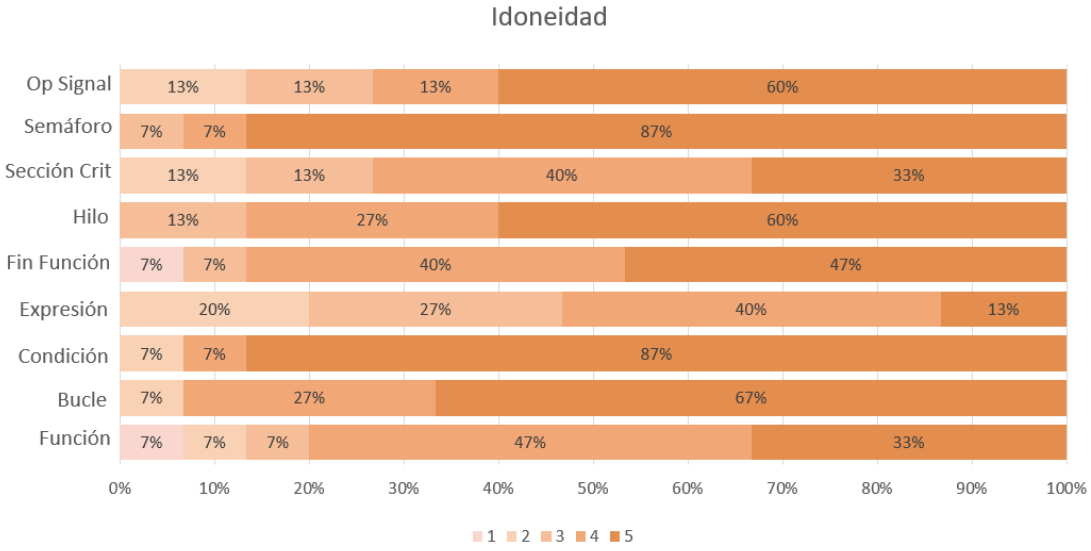


Figura 5.3: Puntuación de idoneidad para cada representación gráfica de la metáfora. La escala de colores representa: 1, poco adecuada a 5, muy adecuada.

Tras estas etapas, se realizó una actividad en la cual los estudiantes debían transcribir y contestar a una serie de preguntas sobre la ejecución de un algoritmo representado con la metáfora que acababan de evaluar. En la mayoría de casos, la transcripción fue correcta o casi correcta, donde el error cometido era mínimo (en el siguiente enlace[2] se adjunta un documento donde pueden revisarse las transcripciones de los estudiantes). En D.1 se proporciona el resultado de la actividad.

En cuanto a las respuestas a las actividades en formato tipo *test*, se puede afirmar que el algoritmo fue comprendido por la mayoría de los participantes. Como se aprecia en la Figura 5.4, las actividades 1, 2, 4, 5 y 6 fueron respondidas correctamente casi por la mayoría de participantes, más de un 70 % (ver C.2, Actividad, para conocer el contenido de las preguntas).

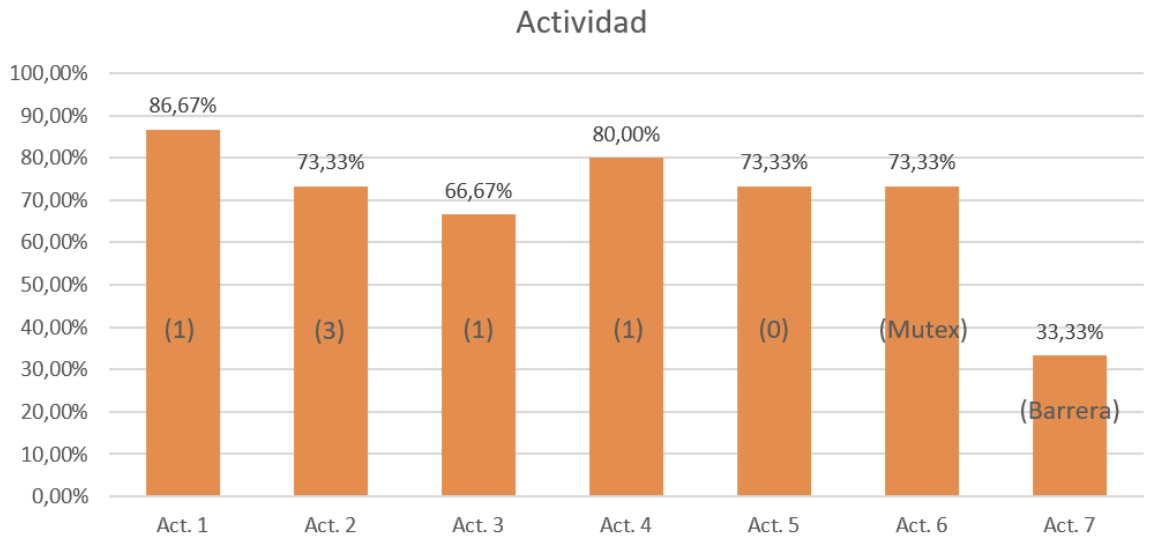


Figura 5.4: Porcentaje de alumnos que respondieron correctamente a cada una de las actividades propuestas. Los números (entre paréntesis) determinan la solución correcta a cada actividad.

Se observa que la cuestión 3 supera el 60 %, pero su resultado debería ser mayor, dado que preguntas similares como 1 y 2 tiene un mayor porcentaje de aciertos. En este cuestión se pregunta cuántos camiones pueden acceder a la vez en la segunda sección crítica. En esencia, solo uno puede acceder a la sección crítica, pues al pasar el vehículo por el tramo de carretera con el pulsador (operación signal) es cuando una entidad da acceso a que otra pueda seguir con su ejecución. Sin embargo, en el vídeo se aprecia como en algún *frame* dos vehículos están sobre la sección crítica, lo cual ha podido confundir a algún estudiante a la hora de responder la cuestión. En cuanto a la pregunta 7, solo el 33 % de los alumnos respondieron bien a esta cuestión, la cual está enfocada a determinar el patrón de sincronización que representa el último semáforo del algoritmo. Esto puede deberse a que, o muchos no conocían

[2] https://pruebasaluuclm-my.sharepoint.com/:t:/g/personal/cristian_gomez2_alu_uclm_es/ESC6brDygmRHrWCUpHOqCe8BIblfb5bga2mcOrNvCrgiqQ?e=bc9s1X

el patrón o simplemente no lo recordaban. Sin embargo, la mayoría estaban de acuerdo en el valor inicial del semáforo y el número de hilos que podían quedarse bloqueados en dicho elemento.

En relación con la motivación intrínseca y el rendimiento subjetivo de la actividad, la Tabla 5.4 muestra los resultados obtenidos de las respuestas de los alumnos. En general, los estudiantes estuvieron involucrados en su ejecución, mostrando interés (INT) y esfuerzo (EFF), al mismo tiempo que se consideraban a ellos mismos más competentes (COM) a la hora de realizar esta tarea. Además, según los resultados, los estudiantes no se encontraron nerviosos (PRE) durante la realización de la prueba. En cuanto a la parte motivadora de la propuesta (PSA), los estudiantes valoraron positivamente la representación definida para modelar algoritmos. Además, sintieron que la metáfora puede ser motivadora para quien está aprendiendo a programar.

MI	Media*	Mediana
INT	4,13 (0.86)	4,00
COM	3,57 (0.72)	4,00
EFF	3,70 (1,05)	4,00
PRE	1,60 (0.91)	1,00
PSA	3,90 (0.92)	4,00

Tabla 5.4: Resultado de las variables que están asociadas a la motivación intrínseca y rendimiento subjetivo. *Se muestra la media y la desviación típica (entre paréntesis).

Respecto a la carga cognitiva de la actividad (ver Tabla 5.5), los alumnos no sintieron que la tarea fuese excesivamente complicada (TD). No obstante, si que mostraron concentración durante su desarrollo (E).

CC	Media*	Mediana
TD	1,83 (1,01)	1,00
E	2,73 (1,36)	2,50

Tabla 5.5: Resultado de las variables que están asociadas a carga cognitiva de la tarea. *Se muestra la media y la desviación típica (entre paréntesis).

Atendiendo a los términos de facilidad de uso (PEOU), utilidad (PU) y futura intención de uso (ITU) de la metáfora, los estudiantes proporcionaron una valoración positiva, la cual se refleja en la Tabla 5.6.

PSC	Media*	Mediana
PEOU	3,93 (0.91)	4,00
PU	4,08 (0.86)	4,00
ITU	3,90 (1,02)	4,00

Tabla 5.6: Resultado de las variables que están asociadas a la percepción subjetiva de la calidad de la propuesta. *Se muestra la media y la desviación típica (entre paréntesis).

Finalmente, con respecto a aquello que más y menos gustó a los participantes sobre la metáfora, se destacan aquellos comentarios más interesantes por su elaboración o repetición.

- **Comentarios positivos**
  - «La metáfora es fácil y sencilla de utilizar».
  - «Dado que el funcionamiento vial real lo conocemos todo el mundo, creo que es una buena práctica para explicar así los algoritmos. Me ha gustado la representación a través de los semáforos y señales».
  - «Me parece una forma muy intuitiva de explicar cómo funciona la ejecución de diversos procesos de forma concurrente para una persona que no tenga unos conocimientos demasiado avanzados en el tema. La representación de los *if*, los bucles y los semáforos me ha resultado especialmente buena».
- **Comentarios negativos**
  - «No considero adecuada la forma en que se apilan los procesos (camiones). Creo que sería más adecuado que se quede uno detrás de otro». Esto puede deberse a que no es una situación natural que los vehículos mientras esperen a acceder a una zona determinada se apilan uno tras otro. Probablemente lo más lógico sería actuar como en la vida real y evitar así posibles malentendidos.
  - «El uso de los túneles para representar distintos tipos de funcionalidades no me parece demasiado adecuada». Este comentario se debe a que esta metáfora también se utiliza, de algún modo, para representar el concepto de la sección crítica, lo cual puede llegar a confundir el uso de una misma metáfora en aspectos distintos.

5.3 Conclusiones

En la sección anterior se describe detalladamente la evaluación realizada y los resultado obtenidos. El objetivo de dicha evaluación era conocer de primera mano si la representación gráfica propuesta, basada en la metáfora de carreteras y señales de tráfico, es útil e idónea para facilitar la comprensión de conceptos y mecanismos relacionados con la programación concurrente.

Los resultados mostrados en las Figuras 5.2 y 5.3 afirman que los elementos mejora evaluados de la metáfora son la representación para el semáforo, condición, hilo, operación signal

y bucle. Sin embargo, aquellas que no resultaron ser tan bien valoradas como las anteriores son la metáfora para la evaluación de expresiones y sección crítica.

Con respecto a los resultados de la actividad, se puede seguir afirmando que la metáfora como método para representar algoritmos es adecuada para que se entienda su propósito. En general, el porcentaje de respuestas correctas para cada pregunta fue elevado, cuyo resultado puede verse en la Figura 5.4.

Lo anteriormente mencionado, además, se puede contrastar con los resultados obtenidos por medio del *framework* TAM, con el cual se midieron variables como la facilidad de uso (PEOU), percepción de utilidad (PU) e intención de uso (ITU). Los resultados arrojados mostraron que los alumnos percibieron la metáfora como un enfoque útil que podría reducir el tiempo necesario para entender un algoritmo. Además, consideran que si recomendarían el uso de esta técnica para representar algoritmos.

En definitiva, las principales conclusiones obtenidas es que la metáfora es fácil y sencilla de utilizar, al mismo tiempo que útil e intuitiva. En parte, este hecho puede asociarse a la relación de correspondencia que existen entre conceptos de circulación vial y términos de programación, ya que los alumnos que comienzan a estudiar contenidos de programación acaban de alcanzar la mayoría de edad y están en situación de obtener la licencia de conducir.

5.4 Análisis de costes

El desarrollo de este proyecto abarca desde enero de 2019 hasta julio de 2019. En este periodo de tiempo se trabajó en el proyecto a tiempo parcial durante 5 días a la semana en el grupo de investigación CHICO. Esto supone un total de 700 horas de trabajo desarrollado aproximadamente. Cabe destacar que en estas horas no se ha incluido el tiempo dedicado en casa y el tiempo empleado para la elaboración del presente documento, lo cual extendería esta estimación.

En este periodo, el autor de este trabajo estuvo contratado como Investigador a Tiempo Parcial en el proyecto de investigación IAPRO[3], cuya función estaba destinada al desarrollo del sistema. El salario obtenido durante estos meses fue entorno a 625€/mes teniendo como categoría «Segunda-O-I».

En el Tabla 5.7 se muestra un desglose de los gastos aproximados durante el desarrollo. Por un lado, se incluyen los gastos de *hardware* para el despliegue del proyecto y, por otro lado, el sueldo del programador teniendo en cuenta lo anteriormente comentado.

[3] `http://blog.uclm.es/grupochico/proyecto-iapro/`

Recurso	Cantidad	Coste
Sueldo programador (700 horas)	1	4.375,00€
Computador personal	1	1.275,00€
Microsoft HoloLens	1	3.299,00€
Total		**8.949,00€**

Tabla 5.7: Desglose de costes del proyecto

5.5 Competencias adquiridas

En este apartado se enumeran y detallan las competencias específicas adquiridas como consecuencia de la realización del presente trabajo.

- **[CE1]**: *Capacidad para la integración de tecnologías, aplicaciones, servicios y sistemas propios de la Ingeniería Informática, con carácter generalista, y en contextos más amplios y multidisciplinares.*

  En este proyecto se integra un dispositivo de RA con el que se visualiza el código fuente de un programa, un *plug-in* externo que ofrece funcionalidades de participación y colaboración remotas y síncronas, así como una multitud de bibliotecas y *frameworks* para atacar diversos problemas.

- **[CE13]**: *Capacidad para utilizar y desarrollar metodologías, métodos, técnicas, programas de uso específico, normas y estándares de computación gráfica.*

  Esta competencia se ha adquirido al utilizar técnicas de modelado para el diseño y creación de modelos 3D, aplicación de materiales y texturas, generación de sombras y creación de animaciones, así como el diseño de escenas e interfaces.

- **[CE14]**: *Capacidad para conceptualizar, diseñar, desarrollar y evaluar la interacción persona-ordenador de productos, sistemas, aplicaciones y servicios informáticos.*

  Con el objetivo de medir la adecuación y comprensión de la metáfora utilizada en el sistema, se ha realizado una evaluación con alumnos, incluyendo preguntas, ejemplos y ejercicios para determinar la utilidad de la propuesta.

- **[CE15]**: *Capacidad para la creación y explotación de entornos virtuales, y para la creación, gestión y distribución de contenidos multimedia.*

  Esta competencia se cumple mediante la creación de un entorno virtual, cuyo propósito es su explotación en el aula al generar visualizaciones dinámicas que ayudan a comprender el flujo de ejecución de un programa.

5.6 Publicaciones científicas

En este apartado se enumeran los trabajos científicos que surgen del desarrollo de este trabajo.

- **Cristian Gmez-Portes**, Santiago Schez-Sobrino, María Á. García, Miguel Á. Redondo, Javier A. Albusac and Manuel Ortega. Aplicación de una metáfora flexible y extensible para la visualización de programas en el contexto del aprendizaje de la programación. *The 21st International Symposium on Computers in Education (SIIE)*, Tomar, Portugal, from 21st to 23rd November 2019.
- Santiago Schez-Sobrino, Maria Á. García, **Cristian Gómez**, David Vallejo, Carmen Lacave, Carlos Glez-Morcillo, Ana I. Molina, Javier A. Albusac and Miguel Á. Redondo. ANGELA: a novel approach of graphic notation based on the metaphor of road signs to facilitate the learning of for E-learning, León, 16-18 de octubre de 2019. programming. *In Proceedinsgs of the Seventh International Conference on Technological Ecosystem for Enhancing Multiculturality*, October 2019, pp.822–829, `https://doi.org/10.1145/3362789.3362871`.
- S. Schez-Sobrino, M.Á. García, **C. Gómez**, C. Glez-Morcillo, D. Vallejo, J.A. Albusac, M.Á. Redondo. Propuesta y evolución multidimensional de una metáfora visual para facilitar el aprendizaje de la programación. *XX Congreso Internacional de Interacción Persona-Ordenador (Interacción 2019)*, San Sebastián, País Vasco (España).
- Santiago Schez-Sobrino, **Cristian Gmez-Portes**, David Vallejo, Carlos Glez-Morcillo, Miguel Á. Redondo.(2019). An Intelligent Tutoring System to Facilitate the Learning of Programming Through the Usage of Dynamic Graphic Visualizations. Artículo entregado para publicación.

5.7 Premios

En esta sección se muestran los premios obtenidos como resultado, tanto del trabajo sobre el que se sustenta este proyecto, es decir, el desarrollo tecnológico realizado en [GP18], como la mejora y evolución descrita en un artículo científico.

- Premio extraordinario Caja Rural al mejor TFG de Castilla-La Mancha en la rama de arquitectura e ingeniería en Toledo, 20 de diciembre de 2018, por el trabajo **AsgAR: Sistema generado de representaciones gráficas para la visualización de programas y algoritmos mediante Realidad Mixta**.
- Premio Jesús Lorés al mejor artículo de investigación presentado en el *XX Congreso Internacional de Interacción Persona-Ordenador (Interacción 2019)* en Donostia, San Sebastián, del 26 de junio al 28 de junio, por el artículo **Propuesta y evolución multidimensional de una metáfora visual para facilitar el aprendizaje de la programación**.

Capítulo 6

Propuesta de negocio

En este capítulo se proporciona una propuesta de negocio sobre una posible explotación de una evolución del producto *software* que este trabajo introduce. En concreto, se describe el modelo de negocio sobre una plantilla estratégica , el plan de trabajo que se llevará a cabo, así como el equipo de personas que se encargarán en desarrollar la infraestructura *software*.

6.1 Modelo de negocio de la propuesta

El estado actual del sistema da pie a que pueda plantearse su explotación en un contexto real, por ejemplo, el aula de una universidad o un centro privado de formación. Como resultado, esta sección describe una posible línea de negocio con la que recuperar la inversión inicial (ver Sección 5.4) y obtener una rentabilidad en un corto periodo de tiempo.

El modelo de negocio a seguir estará basado en licencias. El segmento de clientes deberá estar orientado a universidades, centros educativos de formación profesional o centros de aprendizaje privados. En una primera instancia, las licencias se comprarán anualmente, pudiendo ser renovadas días antes de su finalización. Particularmente, dichas licencias estarán divididas en dos: (1) básica (400,00€ al año) y (2) *premium* (1000,00€ al año) La primera ofrecerá el *software* con la posibilidad de solo visualizar el código fuente de un programa de un modo estático. Por otro lado, la segundo permitirá proporcionar una visualización tanto estática como dinámica (programas multi-hilo y con recursividad). Incluso, dará la opción de integrar todo el contenido que esté siendo actualizado. En una primera etapa comercial se contactará con centros privados para explorar su interés por la herramienta, dado que disponen de más flexibilidad a la hora de adquirir licencias de *software* que las administraciones públicas. Una vez conocida la aceptación, se intentaría integrar la herramienta en diversas universidades para una explotación a gran escala.

En la Figura 6.1 se incluye un lienzo que muestra, de forma descriptiva, las 9 claves que representan un negocio, las cuales son descritas en los siguientes puntos:

- *Propuesta de valor* es el producto o servicio que se ofrece para satisfacer las necesidades de un potencial cliente, en nuestro caso, un producto desarrollado para mejorar el aprendizaje de la programación.

Figura 6.1: *Business Model Canvas*. Plantilla extraída de: *https://assets.strategyzer.com/assets/resources/the-business-model-canvas.pdf*

- *Actividades claves* se refieren a la manera de ejecutar la propuesta de valor. Para este modelo se identifican tres actividades principales: desarrollo, innovación y actividad comercial (contactar con interesados, buscar financiación, etc.).
- *Recursos claves* son los medios necesarios para crear valor para el cliente, por ejemplo, las gafas de RA, la metáfora de carreteras y señales de tráfico o el algoritmo capaz de convertir código fuente a una metáfora de carreteras y señales de tráfico, entre otros.
- *Socios clave* son el conjunto de relaciones o alianzas entre competidores y no competidores. Debido a la dependencia de este *software* con el dispositivo Microsoft HoloLens, la empresa Microsoft sería un socio clave. La UCLM también podría considerarse como otro socio clave, dado que puede contribuir a proporcionar el capital para llevar a cabo tareas de desarrollo e innovación.
- *Relación con clientes* se refiere a un conjunto de clientes que garanticen la supervivencia y éxito del negocio. Dos posibles actividades que podrían retener a potenciales clientes sería ofrecer un mantenimiento gratuito del software y una atención personalizada. Además, para atraer a potenciales clientes se propone la creación de comunidades para que los clientes intercambien ideas, problemas, soluciones, etc.
- *Segmento de clientes* se refiere a la diferenciación de grupos de clientes en base a algún tipo de necesidad. Para este caso se distinguen: centros educativos públicos y/o privados, donde los usuarios objetivos son estudiantes con muy pocos conocimientos en programación.
- *Canales* son los medios por los que se entrega la propuesta de valor. Esto ha sido realizado a través de foros científicos y tecnológicos a nivel nacional y mundial.
- *Fuentes de ingreso* son los medios a través de los cuales se genera flujo de ingresos. En nuestro caso esto se realiza a través de licencias.
- *Estructura de costes*, que describe las consecuencias monetarias mientras opera bajo un modelo de negocio. Esto, en nuestro caso, está relacionado con las nóminas del personal contratado para el desarrollo del proyecto, las infraestructuras *hardware* y *software*, así como el mantenimiento de las relaciones con los socios.

6.2 Plan de trabajo

La Tabla 6.1 muestra la estructura del proyecto a realizar, dividido en un total de 12 meses organizado en **6 paquetes de trabajo** (PT). Resulta importante destacar que los costes asociados a los paquetes de trabajo PT3, PT4 y PT5 están asociados a las nóminas del personal contratado (ver Sección 6.3).

PT1. Gestión del proyecto (M1-M12). Paquete relacionado con la gestión administrativa y técnica del proyecto (adquisición de material informático), generación de informes, gestión de riesgos, etc.; PT2. Elicitación de requisitos y experiencia del usuario (M1-M2). Definición

de los requisitos del proyecto. PT3. Visualización de programas multi-hilo (M2-M7). Paquete de trabajo orientado al desarrollo de un módulo que soporte la visualización dinámica de programas con un enfoque concurrente. PT4. Visualización de programas con recursividad (M7-M10). Diseño e implementación de un módulo para soportar la visualización dinámica de programas con recursividad. PT5. Integración (M7-M12). Desarrollo de las tareas para integrar los módulos diseñados. PT6. Difusión, comunicación y explotación (M1-M12). Paquete que contempla las tareas relacionadas con la asistencia a foros científicos y tecnológicos, así como otro tipo de eventos donde dar visibilidad al producto desarrollado.

Paquete de trabajo	Descripción	Coste estimado (euros)	Inicio (mes)	Fin (mes)
PT1	Gestión del proyecto	18.000,00€	M1	M12
PT2	Elicitación de requisitos	800,00€	M1	M2
PT3	Visualización de programas multi-hilo	35.000,00€	M2	M7
PT4	Visualización de programas con recursividad	21.000,00€	M7	M10
PT5	Integración	21.500,00€	M7	M12
PT6	Difusión, comunicación y explotación.	7.000,00€	M1	M12

Tabla 6.1: Plan de trabajo distribuido en 12 meses

6.3 Composición de equipo

El equipo de trabajo que se compondrá para la evolución del prototipo es la siguiente:

- Director del proyecto. Responsable encargado de gestionar el proyecto al completo, abarcando los paquetes de trabajo PT1, PT2, PT3, PT4, PT5, PT6. Se estima un salario de 2.500,00€ al mes.
- Diseñador. Esta persona será la encargada de las tareas que tienen que ver con un aspecto más creativo, es decir, el diseño de materiales, texturas y modelos, así como la generación de metáforas para el contexto en cuestión, abarcando los paquetes de trabajo PT3, PT4. Particularmente, el candidato valorará la utilización de tecnologías como las siguientes: *Inkscape*, *Gimp*, *Blender*, etc. La necesidad de contar con un profesional de estas características viene en parte por trabajar con tecnología de visualización. Se persigue, por tanto, que los diseños conseguidos involucren aún más al estudiante en el proceso del aprendizaje de la programación. El candidato contará con un contrato a tiempo parcial. Se estima un coste de contratación de 900,00€ al mes.
- Ingeniero de *software*. Será la persona encargada del núcleo principal del desarrollo *software*, participando en los paquetes de trabajo PT2, PT3, PT4, PT5, PT6. El candidato trabajará a tiempo completo en la definición general de una arquitectura de

visualización de programas multi-hilo y recursivos. Para ello, la persona contratada utilizará el motor de juegos Unity y el lenguaje de programación c#. Se pretende, además, desarrollar una metodología que pueda ser usada en cualquier centro educativo, facilitando así su uso por parte de docentes. Se estima un coste de contratación de 1.800,00€ al mes.

- Ingeniero de *software* con perfil en Gráficos por Computador. Esta persona será la encargada de integrar los elementos gráficos generados por el diseñador en el motor de Unity, cuyos paquetes de trabajo son PT3, PT4. El candidato hará uso de algoritmos de Computación Gráfica para añadir un comportamiento personalizado a tales elementos, tratando de crear una experiencia enriquecedora para el estudiante. Este empleado contará con un contrato a tiempo completo. Se estima un coste de contratación de 1.800,00€ al mes.

6.4 Retorno de inversión

Según los costes de la Tabla 6.1, la inversión que se ha de realizar para poner en marcha este proyecto es de 54.400,00€. Claramente, cuantos más clientes adquieran el producto en forma de licencia la inversión se recuperará antes (ver Figura 6.2). Dado que el valor de la licencia básica es de 400,00€ al año y la licencia *premium* 1000,00€ al año, deberían adquirirse entre ambas, a modo de estimación, un total de 54 para recuperar la inversión realizada, lo cual es totalmente factible debido al número de Grados en Ingeniería Informática, Cursos de Formación Profesional y Cursos extraescolares en centros de enseñanza privada.

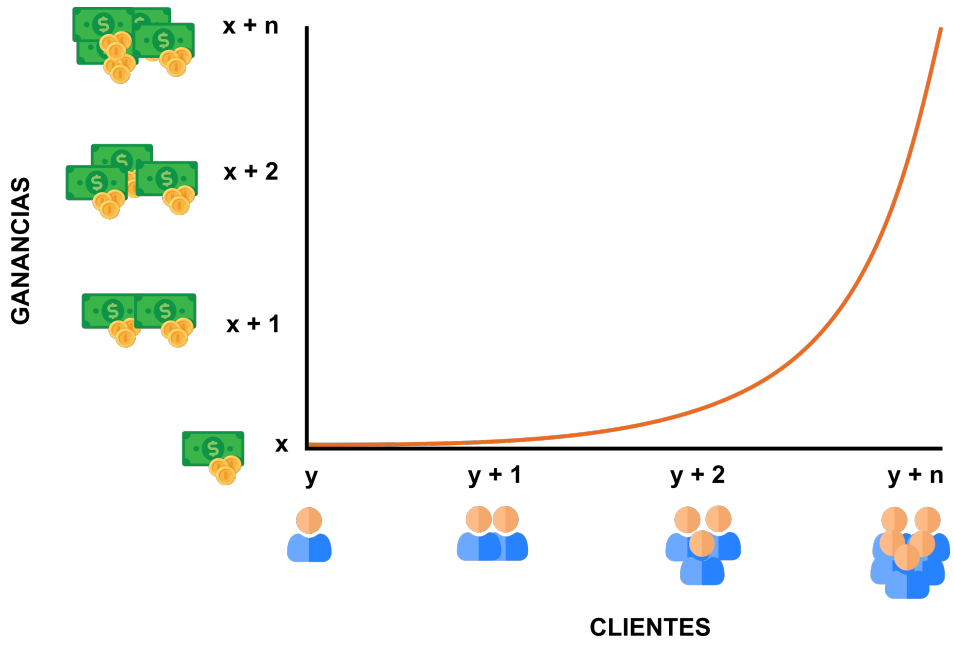


Figura 6.2: Recuperación de inversión

La recuperación de la inversión en un año podría entenderse como una buena aceptación por parte de los centros anteriormente mencionados. Este hecho podría marcar el punto de

partida para expandir la herramienta a nivel europeo, tratando de explorar el interés de los centros extranjeros. En función de su acogida, fundamentalmente en los beneficios obtenidos, y teniendo margen económico para establecer nuevas líneas de negocio, se estudiaría la idea de lanzar las licencias a nivel mundial, con el objetivo de convertir este método de aprendizaje en un estándar para el aprendizaje de la programación.

Capítulo 7

Conclusiones

En este capítulo se recoge brevemente los resultados logrados tras la realización de este TFM, así como a un conjunto de posibles mejoras y propuestas que nacen a partir de este trabajo.

Adicionalmente, se incluye una valoración personal sobre los conocimientos y habilidades adquiridos durante el desarrollo, junto a una reflexión de lo que ha significado su realización y lo que se espera de este proyecto.

7.1 Objetivos alcanzados

En relación con los resultados presentados en el Capítulo 5, se puede afirmar que los objetivos definidos en el Capítulo 2 se han cumplido satisfactoriamente.

Esencialmente, éstos derivan del objetivo general del proyecto, el cual trata de desarrollar e implementar un sistema de visualización dinámica de programas mediante RA para facilitar y mejorar el proceso de aprendizaje de la programación. A continuación, se detalla el grado de consecución de los subobjetivos que han permitido completar el objetivo principal.

- **Propuesta de una metáfora de carreteras y señales de tráfico para soportar la visualización dinámica de programas**. Este subobjetivo trata de modificar y ampliar una metáfora existente desarrollada en un TFG previo para soportar la visualización dinámica de programas. Esta metáfora, en su versión inicial, solo permitía la visualización estática de programas, es decir, representar únicamente la estructura de los programas (p.ej., estructuras de control, declaración de variables, etc.). Esta nueva alternativa contempla dos enfoques. El primero consiste en la modificación de la metáfora de evaluación de expresiones (ver Figura 3.10d) por un diseño basado en un túnel de carreteras, más adaptado e integrado con la misma (ver Figura 4.2). Por otro lado, se ha ampliado la metáfora con dos diseños adecuados perfectamente al conjunto de representaciones que soportan la visualización dinámica de programas (ver Figura 4.2).
- **Diseño y codificación de un sistema de visualización dinámica**. Este subobjetivo pretendía crear un entorno de visualización dinámica de programas. El sistema identifica las sentencias del código fuente, las cuales son convertidas a una metáfora de

carreteras y señales de tráfico, que asocian términos de programación con conceptos de circulación vial. La modalidad dinámica de este sistema es que permite visualizar la ejecución de un programa mediante una animación generada de manera automática, que incluye el movimiento de un vehículo sobre una representación, basada en la metáfora mencionada, simulando ejecutar sentencias. Adicionalmente, este sistema presenta otra singularidad, y es que se caracteriza por ser un enfoque flexible, ya que potencia el uso de otro tipo de paradigmas de programación, en concreto el de la programación concurrente, siendo la metáfora muy adecuada para este tipo de casos.

- **Evaluación y caso de estudio para la programación concurrente y en tiempo real**. En este subobjetivo se ha realizado una evaluación con estudiantes de la asignatura de Sistemas Operativos II, la cual complementa la formación recibida en Sistemas Operativos I y Programación Concurrente y en Tiempo Real con la programación multi-hilo, siendo el objetivo principal medir en términos de comprensión e idoneidad la metáfora propuesta para el paradigma de la programación con varios hilos. Igualmente también se ha medido la modificación del diseño basado en la evaluación de expresiones. El análisis de los resultados demuestra que la notación propuesta ha sido bien recibida por los estudiante, lo que la hace adecuada para representar programas mono-hilo o multi-hilo. Además, se han identificado oportunidades de mejoras que traten de potenciar el enfoque planteado.

7.2 Líneas de trabajo futuras

En esta sección se enumeran un conjunto de evoluciones y propuestas que tratan de mejorar el sistema desarrollado.

Principalmente, uno de los desarrollos con mayor prioridad a tratar es la mejora del sistema para visualizar dinámicamente programas basados en el paradigma de la **programación concurrente**. En este momento, las visualizaciones dinámicas soportan programas con varios procesos, pero únicamente con un solo hilo de ejecución. El objetivo, por lo tanto, se centra en la posibilidad de poder visualizar múltiples hilos simultáneamente, cuyo enfoque, a través de la RA, permita obtener una visión global de todos ellos en ejecución, aparte de hacer un seguimiento personalizado de cada una de las entidades. Una posible solución pasaría por identificar a los procesos como vehículos con un color, de los cuales salgan coches del mismo color que identifiquen a los hilos. Así, se plantea un sistema lo suficientemente flexible para explotar diferentes paradigmas de programación que potencien y faciliten el proceso de aprendizaje de la programación.

A modo de propuesta se plantea, como posible línea de trabajo futura, la visualización dinámica de programas en el contexto de la **recursividad**. Comúnmente, se conocen las dificultades asociadas al aprendizaje de la recursividad, la cual reside, entre otras, en los resultados producidos por cada llamada recursiva, en concreto, el paso de parámetros, paso

de objetos como parámetros, variables locales, la vuelta atrás, etc. Se propone, por tanto, ampliar la metáfora ya existente para crear un entorno basado en un aparcamiento de coches con varios niveles, donde un vehículo se sitúa en la planta superior representando el caso base. Consecuentemente, dicha entidad bajará de nivel o planta a consecuencia de las llamadas recursivas.

Adicionalmente, resulta interesante estudiar el análisis de la explotación del sistema propuesto en este trabajo para entornos de RA sobre **dispositivos móviles**. La idea trata de utilizar *frameworks* de desarrollo actuales que permitan realizar un *tracking* sin marcas en dispositivos iOS y Android, como ARKit y ARCore, respectivamente. Esto supondría explotar el espacio 3D aumentado de un modo económico en términos de costes de dispositivos, puesto que, actualmente, el precio desorbitado de las gafas Microsoft HoloLens impiden su total accesibilidad.

7.3 Conclusión personal

La realización del presente TFM ha supuesto el punto final a mi etapa como estudiante de Máster, cuyo trabajo es fruto de la formación adquirida, la cual me ha permitido potenciar los conocimientos aprendidos durante el Grado.

El hecho de que este trabajo sea parte de un proyecto real dentro de un grupo de investigación me ha servido para desarrollar algunas competencias vistas en el Máster, relacionadas, por ejemplo, con el hecho de comunicar avances del proyecto a públicos especializados, concretamente en congresos internacionales y revistas científicas, así como la planificación y coordinación y gestión del mismo.

Aparte, también se han adquirido conocimientos profundos en tecnologías modernas y punteras que han dado lugar a la integración de información virtual en escenarios reales, o incluso en la comunicación entre aplicaciones alojadas en computadores separados físicamente.

En definitiva, este proceso me ha servido para crecer en lo profesional, pero también en lo personal, pues ha sido un periodo de aprendizaje enriquecedor. Además, en cuanto al proyecto en cuestión, espero que este trabajo no caiga en el olvido y pueda seguir evolucionando para resolver el principal problema por el que fue planteado, el aprendizaje de la programación.

ANEXOS

Anexo A

Código fuente del proyecto

A.1 Cliente

La aplicación del cliente, es decir, aquella encargada de visualizar el código fuente de un programa mediante una metáfora de carreteras y señales de tráfico, presenta la siguiente estructura de directorios:

- **Assets**: contiene todos los recursos utilizados en el proyecto.
  - **Animations**: contiene las animaciones utilizadas por los modelos.
  - **Animator**: contiene las máquinas de estados que controlan las animaciones usadas por un objeto.
  - **External**: contiene las bibliotecas utilizadas para el desarrollo de la aplicación.
  - **Models**: almacena los modelos 3D generados con la herramienta Blender.
  - **Plugins**: contiene los componentes externos usados que extienden las capacidades de la aplicación.
  - **Prefabs**: contiene los objetos construidos en las escenas del proyecto con valores y atributos específicos.
  - **Resources**: contiene los recursos que son creados en tiempo de ejecución.
  - **Scenes**: contiene las escenas de la aplicación.
  - **Scripts**: contiene el código fuente C# de la aplicación.
  - **Shaders**: contiene los programas *vertex y fragment shader* aplicados a los modelos de la aplicación.
  - **UI**: almacena los modelos utilizados por la IU de la aplicación.
- **ProjectSettings**: contiene la configuración del entorno Unity 3D.

El código fuente relativo a esta parte del proyecto puede encontrarse en el repositorio de Bitbucket con el siguiente enlace: `https://bitbucket.org/Cristiancgp/tfm-asgar/src/master/`

A.2 Servidor

En cuanto a la aplicación del servidor, es decir, aquella que se ejecuta en el entorno de Eclipse, presenta la siguiente estructura de directorios:

- **META-INF**: almacena los metadatos con el contenido del plug-in.
- **lib**: contiene las bibliotecas utilizadas en el desarrollo del plug-in.
- **src**: contiene el código fuente en Java del plug-in de Eclipse.

El código fuente relativo a esta parte del proyecto puede encontrarse en el repositorio de Bitbucket con el siguiente enlace: `https://bitbucket.org/chico_uclm/collece-2.0-alvis-plugin/src/master/`

Anexo B

Indicadores del curso 2018-2019

Conjunto de indicadores del curso 2018-2019 del Grado y Máster que se imparten en la Escuela Superior de Informática de Ciudad Real.

Indicadores Curso 2018/19 (definitiva)

Resultados académicos generales 2017/18 (Histórico aprobados en grado 62,61,64)

	% Aprobados			% Suspensos			% no presentados		
	16/17	**17/18**	**18/19**	**16/17**	**17/18**	**18/19**	**16/17**	**17,18**	**18/19**
GRADO	63,76	58,93	64,68	20,47	24,4	22,12	15,76	16,67	13,20
MÁSTER	88,59	78,40	64,29	1,34	9,55	2,14	10,07	11,06	33,57

Evolución de titulados

	13/14	**14/15**	**15/16**	**16/17**	**17/18**	**18/19**
GRADO	28	37	61	63	58	83
MÁSTER	2	5	9	3	12	11

Evaluación número de alumnos

	Nuevo ingreso / Totales matriculados						
	13/14	**14/15**	**15/16**	**16/17**	**17/18**	**18/19**	**19/20**
GRADO	111/505	124/ 549	134/574	142/604	151/609	156/621	157/627
MÁSTER	21/21	15/31	17/33	15/30	22/41	12/33	14/22

Tasa de abandono (no matriculados en el N ni N+1 año Vs nuevo ingreso)

	11/12	**12/13**	**13/14**	**14/15**	**15/16**	**16/17**
GRADO (35)	43,48	36,94	33,33	33,59		
MÁSTER (12)		9,09	52,63	42,86	31,25	26,67

Tasa de graduación (graduados en el N o N+1 año Vs nuevo ingreso)

	11/12	12/13	13/14	14/15	15/16	16/17
GRADO (30)	23,48	22,52	25.00	30,53		
MÁSTER (85)		54,55	21,05	42,86	62.50	66,67

Tasa de eficiencia (créditos matriculados vs créditos del plan)

	13/14	14/15	15/16	16/17	17/18	18/19
GRADO (85)	95,43	90,60	85,67	84,30	82,08	79,57
MÁSTER (85)	94,74	96,15	95,74	100	84,91	99,40

Tasa de rendimiento (créditos superados Vs créditos matriculados)

	13/14	14/15	15/16	16/17	17/18	18/19
GRADO	62,46	61,65	64,27	64,24	59,24	65,73
MÁSTER	73,80	72,44	80,93	86,98	79,53	77,62

Tasa de éxito (créditos superados Vs créditos presentados)

	13/14	14/15	15/16	16/17	17/18	18/19
GRADO	76,16	74,74	76,90	76,44	71,57	75,72
MÁSTER	91,85	96,58	98,86	98,24	90,18	91,74

Anexo C

Modelo de encuesta para programación concurrente

Encuesta realizada a los estudiantes de la asignatura de Sistemas Operativos II (SSOOII) para evaluar la efectividad de la notación en el contexto de la programación concurrente.

C.1 Pre-test

Primera prueba en la que los estudiantes debían especificar su opinión, responder a cuestiones o puntuar su nivel de conocimiento en cuanto a tareas de programación se refiere.

C.2 Post-test

Segunda prueba en la que los estudiantes debían valorar las dimensiones especificadas en una escala de Likert de 5 puntos para valorar objetivamente la notación. Por otro lado, se incluye otra fase en la que deben de resolver una tarea que consiste en transcribir un programa visualizado en forma de vídeo a pseudocódigo o el lenguaje que ellos prefieran. Además, se añaden una serie de preguntas acerca de la ejecución del algoritmo visualizado para valorar la comprensión de la notación. Finalmente, se incluyen un conjunto de cuestiones que tratan de valorar las dimensiones de la notación.

ID (Las cuatro últimas cifras de tu DNI y letra)

Pre-test

Gracias por participar en esta experiencia, que tiene como objetivo evaluar una notación gráfica (basada en la metáfora de la señalización de las carreteras) para representar algoritmos.

La información recogida en esta prueba, así como tus datos personales, serán tratados con confidencialidad y serán utilizados exclusivamente para este estudio.

La información recogida en esta prueba será tratada con absoluta confidencialidad y será utilizada exclusivamente para este estudio.

LEE ATENTAMENTE EL ENUNCIADO DE CADA CUESTIÓN ANTES DE CONTESTAR. SI TIENES ALGUNA DUDA LLAMA A ALGUNO DE LOS ORGANIZADORES DE LA ACTIVIDAD.

Sexo		**Edad**	**Repite Curso**	**Tipo de Acceso al Grado**		
Masculino	*Femenino*		*Si/No*	*FP*	*Bachillerato*	*Otro*

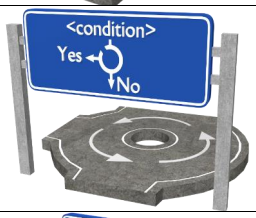
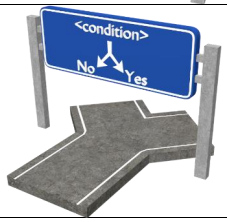
Marca la respuesta que mejor describe tu opinión, teniendo en cuenta que 1 significaría "Totalmente en desacuerdo", y 5, "Totalmente de acuerdo".		**Rodea con un círculo**				
CPAS1.	Encuentro frustrante programar.	1	2	3	4	5
CPAS2.	Programar es una actividad que me diferencia del resto.	1	2	3	4	5
CPAS3.	Creo que la programación no es necesaria.	1	2	3	4	5
CPAS4.	Siempre me esfuerzo por mejorar los programas que escribo.	1	2	3	4	5
CPAS5.	La programación es aburrida.	1	2	3	4	5
CPAS6.	La programación facilita la vida de la humanidad.	1	2	3	4	5
CPAS7.	Si durante la escritura de un programa encuentro un problema que no puedo resolver a corto plazo, no me doy por vencido/a hasta que lo resuelvo.	1	2	3	4	5
CPAS8.	Me pongo nervioso/a cuando tengo que programar.	1	2	3	4	5
CPAS9.	Las actividades relacionadas con la programación me ponen nervioso/a.	1	2	3	4	5
CPAS10.	Trato de investigar para ser un buen programador/a.	1	2	3	4	5
CPAS11.	Cuando comienzo a programar me siento realmente aburrido/a.	1	2	3	4	5
CPAS12.	La programación mejora las habilidades para resolver problemas.	1	2	3	4	5
CPAS13.	Siempre que empiezo a trabajar en un programa, trato de acabarlo antes de hacer cualquier cosa.	1	2	3	4	5
CPAS14.	La sola idea de programar me pone nervioso/a.	1	2	3	4	5
CPAS15.	Creo que muchos de los problemas que no se han resuelto hasta ahora se podrán resolver gracias a los avances en programación.	1	2	3	4	5
CPAS16.	Sigo la evolución de los avances en programación.	1	2	3	4	5

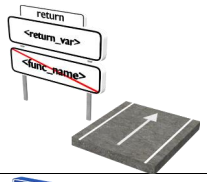
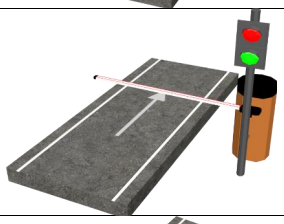
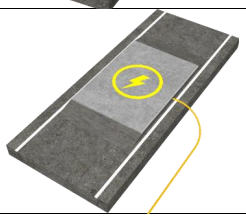
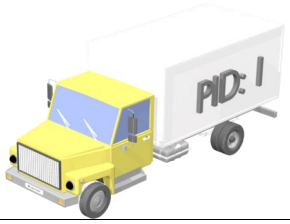
CPKL. Puntúa tu **nivel de conocimiento** en los siguientes aspectos, teniendo en cuenta *que 1 representa "Muy poco", y 5, "Mucho*":	**Rodea con un círculo**
CPKL1. Programación.	1 2 3 4 5
CPKL2. Programación en lenguaje C.	1 2 3 4 5
CPKL3. Programación concurrente.	1 2 3 4 5

ID (Las cuatro últimas cifras de tu DNI)

Post-test

Puntúa la facilidad de entender y la idoneidad de los elementos de la notación propuesta en una escala de 1 a 5 (siendo *1* la *menor valoración* y *5* la *mayor*). Rodea con un círculo tu valoración

Elemento	**Representación**	**Fácil de entender** (Consideras que la representación elegida es *fácil de entender*)	**Idoneidad** (Consideras que la representación elegida es *adecuada* para representar el código del que se trata de abstraer)
Definición de función		1 2 3 4 5	1 2 3 4 5
Sentencia de bucle		1 2 3 4 5	1 2 3 4 5
Sentencia de condición		1 2 3 4 5	1 2 3 4 5

Retorno de función		1 2 3 4 5	1 2 3 4 5
Evaluación de expresiones		1 2 3 4 5	1 2 3 4 5
Sección crítica		1 2 3 4 5	1 2 3 4 5
Semáforo		1 2 3 4 5	1 2 3 4 5
Operación Signal		1 2 3 4 5	1 2 3 4 5
Proceso		1 2 3 4 5	1 2 3 4 5

¿Se te ocurre alguna alternativa a la representación anterior para visualizar el concepto de mandar una señal a un semáforo?

Actividad. La actividad a realizar consiste en reproducir el siguiente vídeo y responder a las preguntas.

El vídeo muestra la ejecución de un programa multi-hilo atendiendo a las representaciones gráficas que has valorado en la sección anterior.

Debes asumir el valor inicial para las siguientes variables:

count <- 0
n <- 4

https://www.youtube.com/watch?v=66zaRSIjbPA

¿Podrías transcribir el programa que se ha ejecutado en el vídeo a pseudocódigo?

Act. 1. Elige una de las siguientes respuestas enumeradas entre 0 y 4, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**				
¿Cuántos camiones acceden, a la vez, a la primera sección crítica?	0	1	2	3	4

Act. 2. Elige una de las siguientes respuestas enumeradas entre 0 y 4, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**				
¿Cuántos camiones pueden quedarse bloqueados como máximo en el segundo semáforo?	0	1	2	3	4

Act. 3. Elige una de las siguientes respuestas enumeradas entre 0 y 4, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**				
¿Potencialmente, cuántos camiones pueden acceder, a la vez, en la segunda sección crítica?	0	1	2	3	4

Act. 4. Elige una de las siguientes respuestas enumeradas entre 0 y 1, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**

¿Cuál es el valor inicial del primer semáforo?	0 1

Act. 5. Elige una de las siguientes respuestas enumeradas entre 0 y 1, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**
¿Cuál es el valor inicial del segundo semáforo?	0 1

Act. 6. Elige una de las siguientes respuestas, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**
¿Cuál sería un nombre adecuado para el primer semáforo?	Mutex Barrera Interruptor Marcador

Act. 7. Elige una de las siguientes respuestas, teniendo en cuenta que solo una de ellas es correcta.	**Rodea con un círculo**
¿Cuál sería un nombre adecuado para el segundo semáforo?	Mutex Barrera Interruptor Marcador

Valoración de dimensiones. Tras realizar la actividad, completa las siguientes tareas.

MI. Puntúa las siguientes afirmaciones de 1 a 5, *teniendo en cuenta que 1 significaría "Totalmente en desacuerdo", y 5, "Totalmente de acuerdo".*	**Rodea con un círculo**				
INT1. Esta actividad me ha resultado **divertida**.	1	2	3	4	5
INT2. Esta actividad me ha resultado **interesante**.	1	2	3	4	5
COM1. Creo que **he realizado bien** esta actividad.	1	2	3	4	5
COM2. **Soy bueno** realizando **este tipo de actividades**.	1	2	3	4	5
EFF1. Me he **esforzado** en hacer bien esta actividad.	1	2	3	4	5
EFF2. Era **importante para mí** resolver bien esta actividad.	1	2	3	4	5
PRE1. Me he sentido **nervioso** mientras realizaba esta actividad.	1	2	3	4	5

CLT. Puntúa las siguientes afirmaciones de 1 a 5, *teniendo en cuenta que 1 significaría "Totalmente en desacuerdo", y 5, "Totalmente de acuerdo".*	**Rodea con un círculo**				
TD1. La **tarea** a realizar (comprensión del algoritmo en la representación propuesta) es **difícil**.	1	2	3	4	5
TD2. La **tarea** a realizar requiere mucho **esfuerzo mental**.	1	2	3	4	5
E1. He estado muy **concentrado** resolviendo la actividad.	1	2	3	4	5
E2. He tenido que **esforzarme** bastante para entender el algoritmo con la notación propuesta.	1	2	3	4	5

PSA. Puntúa las siguientes afirmaciones de 1 a 5, *teniendo en cuenta que 1 representa "Totalmente en desacuerdo", y 5, "Totalmente de acuerdo".*	**Rodea con un círculo**				
PSA1. Creo que la notación propuesta puede ser **motivadora** para quien está aprendiendo a programar.	1	2	3	4	5
PSA2. **Me gusta** la representación propuesta para modelar algoritmos.	1	2	3	4	5

En relación a la representación mostrada para enseñar el funcionamiento de algoritmos, puntúa las siguientes cuestiones de 1 a 5, *teniendo en cuenta que 1 representa "Totalmente en desacuerdo", y 5, "Totalmente de acuerdo".*	**Rodea con un círculo**				
PEOU1. Diseñar algoritmos usando esta notación resulta **simple** y **fácil**	1	2	3	4	5
PEOU2. Me ha resultado **fácil entender** lo que la visualización mostrada pretendía representar	1	2	3	4	5
PEOU3. Este método de representación es **fácil de aprender**	1	2	3	4	5
PU1. Creo que este método de representación podría **reducir el tiempo** necesario **para entender** un algoritmo	1	2	3	4	5
PU2. En general, considero que este método de representación es útil.	1	2	3	4	5
PU3. Creo que este método de representación es **útil** para entender el funcionamiento de un algoritmo	1	2	3	4	5
PU4. Creo que las **representaciones** en esta notación son **organizadas, claras, concisas y no ambiguas**	1	2	3	4	5
PU5. Creo que este método de representación es lo **suficientemente expresivo** para representar los requisitos de un algoritmo	1	2	3	4	5
PU6. En general, creo que este método de representación proporciona un método **efectivo** para **describir algoritmos**	1	2	3	4	5
PU7. Usar este método de modelado **podría ayudarme** a explicar el funcionamiento de un algoritmo	1	2	3	4	5
ITU1. Si tengo que explicar el funcionamiento de algoritmos en el **futuro**, **me gustaría utilizar este método** de representación	1	2	3	4	5
ITU2. Sería **fácil** para mí **adquirir destreza** en el uso de esta representación	1	2	3	4	5
ITU3. **Tengo la intención de utilizar** este método de representación **en el futuro**	1	2	3	4	5
ITU4. **Recomendaría** el uso de esta técnica de representación de algoritmos para explicar programación	1	2	3	4	5

Anexo D

Actividad a resolver por los alumnos

Algoritmo resuelto de la actividad propuesta en C.2 a los estudiantes de SSOOII.

```
n = number of threads
count = 0
mutex = 1
barrier = 0

mutex.wait()
    count += 1
mutex.signal()

if count == n:
    barrier.signal()

barrier.wait()
barrier.signal()

####################
# Critical section #
####################
```

Listado D.1: Pseudocódigo que representa el patrón de sincronización «barrera»

Anexo E

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Referencias

[AICG11] Silvia Abrahao, Emilio Insfran, José Angel Carsí, and Marcela Genero. Evaluating requirements modeling methods based on user perceptions: A family of experiments. *Information Sciences*, 181(16):3356–3378, 2011.

[Ans15] Andrus Ansip. Digital skills, jobs and the need to get more europeans online. `https://ec.europa.eu/commission/commissioners/2014-2019/ansip/blog/digital-skills-jobs-and-need-get-more-europeans-online_en`, 2015. Última consulta en 15/06/2019.

[Azu97] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[BABL+11] Mordechai Ben-Ari, Roman Bednarik, Ronit Ben-Bassat Levy, Gil Ebel, Andrés Moreno, Niko Myller, and Erkki Sutinen. A decade of research and development on program animation: The jeliot experience. *Journal of Visual Languages & Computing*, 22(5):375–384, 2011.

[BBF+14] Jorge Bacca, Silvia Baldiris, Ramon Fabregat, Sabine Graf, et al. Augmented reality trends in education: a systematic review of research and applications. 2014.

[CCP02] Kar Wee Chia, Adrian David Cheok, and Simon JD Prince. Online 6 dof augmented reality registration from natural features. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 305–313. IEEE, 2002.

[CO15] Ibrahim Cetin and M Yasar Ozden. Development of computer programming attitude scale for university students. *Computer Applications in Engineering Education*, 23(5):667–672, 2015.

[Coc08] Alistair Cockburn. Using both incremental and iterative development. *STSC CrossTalk (USAF Software Technology Support Center)*, 21(5):27–30, 2008.

[DA12] SRM Derus and Ahmad Zamzuri Mohamad Ali. Difficulties in learning programming: Views of students. In *1st International Conference on Current Issues in Education (ICCIE 2012)*, pages 74–79, 2012.

[Dav89] Fred D Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.

[DC02] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):932–946, 2002.

[DD14] Matt Dunleavy and Chris Dede. Augmented reality teaching and learning. In *Handbook of research on educational communications and technology*, pages 735–745. Springer, 2014.

[Dis16] Lydia Dishman. Why coding is still the most important job skill of the future. *Fast Company*, 14, 2016.

[DLRTH11] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. Understanding the syntax barrier for novices. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 208–212. ACM, 2011.

[DSKEB15] Phil Diegmann, Manuel Schmidt-Kraepelin, Sven Eynden, and Dirk Basten. Benefits of augmented reality in educational environments-a systematic literature review. *Benefits*, 3(6):1542–1556, 2015.

[Fia05] Mark Fiala. Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596. IEEE, 2005.

[GP18] Cristian Gómez Portes. Asgar: sistema generador de representaciones gráficas para la visualización de programas y algoritmos mediante realidad mixta. Trabajo fin de grado, Universidad de Castilla-La Mancha, 2018. Disponible en `https://ruidera.uclm.es/xmlui/bitstream/handle/10578/19067/TFG_Cristian%20G%c3%b3mez%20Portes.pdf?sequence=1&isAllowed=y`.

[GPM18] Francisco José García-Peñalvo and Antonio José Mendes. Exploring the computational thinking effects in pre-university education. *Computers in Human Behaviour*, 80:407–411, 2018.

[GVAC13] Carlos González, David Vallejo, Javier Albusac, and José Jesús Castro. *Realidad aumentada. Un enfoque práctico con ARToolkit y Blender*. Identic, Ciudad Real, 2013.

[HDS02] Christopher D Hundhausen, Sarah A Douglas, and John T Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

[JD02] Frédéric Jurie and Michel Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):996–1000, 2002.

[JDGCGA12] Guillermo Jimenez-Diaz, Pedro A Gonzalez-Calero, and Mercedes Gomez-Albarran. Role-play virtual worlds for teaching object-oriented design: the virplay development experience. *Software: Practice and Experience*, 42(2): 235–253, 2012.

[KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pages 85–94. IEEE, 1999.

[KM06] Päivi Kinnunen and Lauri Malmi. Why students drop out cs1 course? In *Proceedings of the second international workshop on Computing education research*, pages 97–108. ACM, 2006.

[KST01] Colleen Kehoe, John Stasko, and Ashley Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[LGPMV17] Faraón Llorens, Francisco J García-Peñalvo, Xavier Molero, and Eduardo Vendrell. La enseñanza de la informática, la programación y el pensamiento computacional en los estudios preuniversitarios. *Education in the Knowledge Society (EKS)*, 18(2):7–17, 2017.

[LL+15] Faraón Llorens Largo et al. Dicen por ahí... que la nueva alfabetización pasa por la programación. 2015.

[LLF05] Vincent Lepetit, Pascal Lagger, and Pascal Fua. Randomized trees for real-time keypoint recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 775–781. IEEE, 2005.

[LS87] Jill H Larkin and Herbert A Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, 11(1):65–100, 1987.

[Mar07] David Marimon. Advances in top-down and bottom-up approaches to video-based camera tracking. 2007.

[min19] Inserción laboral de los egresados universitarios - curso 2013-2014 (análisis hasta 2018). `http://www.ciencia.gob.es/stfls/MICINN/Prensa/FICHEROS/2018/190704-Informe_Laboral.pdf`, 2019. Última consulta en 28/12/2019.

[MOM18] Aman Shankar Mathur, Burcu Kulahcioglu Ozkan, and Rupak Majumdar. idea: an immersive debugger for actors. In *Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang*, pages 1–12. ACM, 2018.

[MR04] Iain Milne and Glenn Rowe. Ogre: Three-dimensional program visualization for novice programmers. *Education and Information Technologies*, 9(3):219–237, 2004.

[MTUK95] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies*, volume 2351, pages 282–293. International Society for Optics and Photonics, 1995.

[Mye90] Brad A Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.

[NF02] Leonid Naimark and Eric Foxlin. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 27. IEEE Computer Society, 2002.

[NNB04] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1. IEEE, 2004.

[NRA+02] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *ACM Sigcse Bulletin*, volume 35, pages 131–152. ACM, 2002.

[PC08] Antonio Pérez Carrasco. Srec: sistema educativo para la animación de la recursividad. Trabajo fin de grado, Universidad Rey Juan Carlos, 2008. Disponible en `https://eciencia.urjc.es/bitstream/handle/10115/3446/Memoria_PFC.pdf?sequence=1&isAllowed=y`.

[PC11] Antonio Pérez Carrasco. Sistemas generador de animaciones interativas para la docencia de algoritmos recursivos. Tesis de doctorado, Universidad Rey Juan Carlos, 2011. Disponible en `https://pdfs.semanticscholar.org/accc/abd6f60649385914936599111186fd1fce729.pdf`.

[PC13] Martinha Piteira and Carlos Costa. Learning computer programming: study of difficulties in learning programming. In *Proceedings of the 2013 Inter-*

national Conference on Information Systems and Design of Communication, pages 75–80. ACM, 2013.

[PMB10] Jan L Plass, Roxana Moreno, and Roland Brünken. *Cognitive load theory*. Cambridge University Press, 2010.

[RC08] Jeffrey Rubbin and Dana Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008.

[RT00] Glenn Rowe and Gareth Thorburn. Vince—an on-line tutorial tool for teaching introductory programming. *British Journal of Educational Technology*, 31(4):359–369, 2000.

[SB02] Gilles Simon and M-O Berger. Pose estimation for planar structures. *IEEE Computer Graphics and Applications*, 22(6):46–53, 2002.

[SL04] Iryna Skrypnyk and David G Lowe. Scene modelling, recognition and tracking with invariant image features. In *Third IEEE and ACM international symposium on mixed and augmented reality*, pages 110–119. IEEE, 2004.

[SS12] Teemu Sirkiä and Juha Sorva. Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, pages 19–28. ACM, 2012.

[VIHLPV17] J Ángel Velázquez-Iturbide, Isidoro Hernán-Losada, and Maximiliano Paredes-Velasco. Evaluating the effect of program visualization on student motivation. *IEEE Transactions on Education*, 60(3):238–245, 2017.

[VLF04] Luca Vacchetti, Vincent Lepetit, and Pascal Fua. Stable real-time 3d tracking using online and offline information. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1385–1391, 2004.

[Win06] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[Win08] Jeannette M Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[YLHC15] Jeong Yang, Young Lee, David Hicks, and Kai H Chang. Enhancing object-oriented programming education using static and dynamic visualization. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2015.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfm** (versión 0.20200211) que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`