Progamer: aprendiendo a programar usando videojuegos como metáfora para visualización de código.

J.J. Asensio, A.M. Mora,
P. García-Sánchez, J.J. Merelo

Departamento de Arquitectura y Tecnología de Computadores.
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación.
Universidad de Granada, España
{asensio, amorag, pablogarcia, jmerelo}@ugr.es

Resumen Las herramientas de visualización de software a nivel de componente pueden ser utilizadas durante el desarrollo y mantenimiento del mismo para detectar y visualizar posibles defectos. Estas herramientas suelen ser diseñadas como plugins en los entornos de programación y muestran las relaciones entre las clases permitiendo navegar también visualmente a través del software. Sin embargo también pueden ser útiles durante el aprendizaje de los conceptos básicos de programación. En este sentido es necesario dotar a las herramientas de características adicionales que faciliten este aprendizaje. Para tal fin, este trabajo explora el uso de juegos serios y técnicas de gamificación presentando la herramienta Progamer. Progamer es un plugin de Eclipse para visualización de código Java a nivel de método, basado en mapa, que utiliza los videojuegos de plataformas como metáfora conceptual.

1. Introducción

La visualización de software es un amplio área de investigación que cubre las diferentes técnicas de asistencia empleadas en las actividades de ingeniería del software, tales como especificación, diseño, programación, test, mantenimiento, ingeniería inversa o re-ingeniería. Los métodos empleados, representan gráficamente (en 2D o 3D), de forma estática o dinámica (con animaciones), información relacionada con la estructura, tamaño, historia o comportamiento de un sistema software. El objetivo es facilitar la comprensión de diferentes aspectos del sistema. Esta visualización también permite analizar complejos sistemas y detectar anomalías o defectos de calidad en el software, lo que facilita su evolución. Por otra parte, a nivel educativo, cabe destacar su especial utilidad para el aprendizaje de los conceptos de programación y algoritmos básicos.

Para representar la estructura del software (generalmente orientado a objetos), normalmente, el enfoque consiste en visualizar las dependencias existentes entre las diferentes clases. Estas dependencias puede ser modeladas como un grafo por lo que la visualización en forma de grafo resulta adecuada. El software a

un nivel más básico, es decir, a nivel de método, puede ser también representado de esta forma, mediante un diagrama de flujo. Este diagrama está compuesto por nodos y enlaces.

Sin embargo, el código de un método cumple además con las reglas sintácticas del lenguaje, por lo que también puede ser representado mediante su árbol sintáctico abstracto. En este árbol los nodos representan los diferentes elementos del lenguaje utilizados y las aristas su relación de inclusión (por ejemplo el nodo IF tiene como nodos hijos la parte THEN y la parte ELSE, cada una con sus propios nodos hijos). A diferencia del diagrama de flujo, la visualización de un árbol es mucho más flexible. La jerarquía de un árbol puede ser visualizada mediante un mapa 2D sin necesidad de utilizar flechas [1].

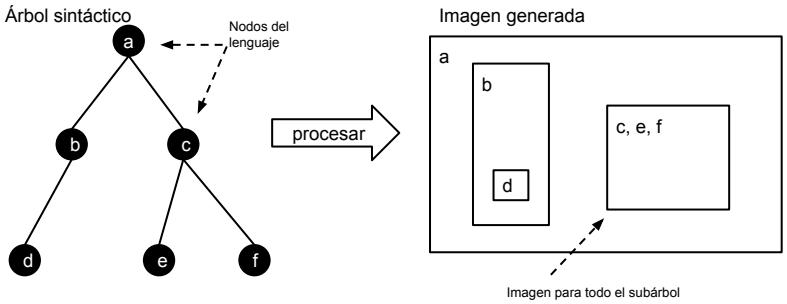

Figura 1. La imagen está compuesta por los gráficos generados para los elementos a, b, c y d. Los nodos e y f comparten representación con el nodo c. La disposición de unas imágenes dentro de otras siguen unas reglas de composición determinadas.

La representación en forma de mapa, permite visualizar los diferentes nodos del lenguaje empleados con un nivel de precisión arbitrario. Es decir si por ejemplo no queremos visualizar detalles de una expresión, podemos visualizarla como un determinado objeto gráfico del mapa sin mostrar en su interior los elementos que la componen (ver figura 1).

El establecimiento de reglas adecuadas para la representación de los diferentes nodos del lenguaje y su composición interna (nodos hijos) nos permite proyectar de manera flexible pero completamente objetiva, el código del programa sobre una imagen representativa. Esta flexibilidad creativa que ofrecen las reglas de composición, habilitan la cosificación o materialización de las metáforas conceptuales [2] en imágenes que pueden ser generadas automáticamente según qué aspecto del código se quiera reflejar con ellas. De esta forma es posible visualizar un mismo programa, usando diferentes metáforas, atendiendo al contexto de utilidad o asistencia deseada, como puede ser mantenimiento, evolución, tests, calidad, navegación, etc.

En el caso del aprendizaje, interesa que la metáfora utilizada resulte atractiva para el estudiante. Para ello, este trabajo propone utilizar el escenario de un videojuego de plataformas como metáfora del árbol sintáctico abstracto del programa. Esta visualización encaja bien dentro de las técnicas de gamificación en educación como las empleadas en [3], donde se gamifica todo el contexto de enseñanza de una asignatura de programación. Por otra parte, resulta ser también una visualización dinámica, puesto que la metáfora empleada es animada y existe una asociación directa entre ambos contextos semánticos, es decir, el protagonista del videojuego corre por el mapa y el ordenador ejecuta (corre) el programa. Esta metáfora también permite la creación de juegos serios donde la mecánica en realidad sería muy distinta a la del videojuego original, pues se trataría de diseñar un mapa (con posibles restricciones) que al ser recorrido resuelva el problema dado.

Para la implementación de una herramienta de visualización dinámica de estas características se ha utilizado el lenguaje Java y el entorno Eclipse. El resto de este trabajo está organizado de la siguiente forma: en la sección 2 se explora el contexto taxonómico, se discute el concepto de legibilidad del código y su relación con algunas herramientas existentes para el aprendizaje de programación orientada a objetos, se describen las técnicas de gamificación y la importancia de su papel en la metáfora conceptual propuesta. En la sección 3 se describe la herramienta presentada concretando sus características. En la sección 4 se detallan los principales aspectos de la implementación. Y por último en la sección 5 se establecen las conclusiones obtenidas y la continuación de este trabajo junto con los agradecimientos.

2. Contexto

2.1. Metáfora conceptual

La idea de utilizar metáforas para la representación y comprensión de conceptos abstractos es fundamental durante el aprendizaje. El término "metáfora"que se usa aquí va más allá del sentido lingüístico habitual utilizado como recurso literario. En realidad se refiere al modelo metafórico por el cual se estructura el conocimiento de un dominio (objetivo) mediante la asociación al mismo de conceptos y relaciones de otro dominio (fuente) existente con el que ya se está familiarizado. En [2] se explora el concepto de metáfora en el contexto computacional.

En matemáticas se utilizan metáforas a menudo para enseñar todo tipo de conceptos. Algunos símbolos matemáticos, como por ejemplo, los paréntesis, pueden ser interpretados como metáforas, en el sentido de que encierran explícitamente un determinado contenido. Podríamos decir que su uso como símbolo posee un significado visual añadido. Existe sin embargo un inconveniente cuando se hace un uso intensivo de estos símbolos ya que pierden su capacidad explicativa, como se observa a continuación en el siguiente programa escrito en Lisp:

```
(defun factorial (n) (if (<= n 1) 1 (* n (factorial (- n 1)))))
```

Otro ejemplo es el uso de flechas para indicar transiciones de un punto a otro. Al igual que con los paréntesis, al aumentar el número de flechas, aumenta la dificultad de comprensión del diagrama. Además, al ser utilizadas frecuentemente en diferentes contextos y con diferente significado, las flechas adquieren un cierto nivel de abstracción, que devalúa todavía más su capacidad aclaratoria.

Con estos ejemplos se intenta ilustrar la importancia y necesidad de encontrar representaciones visuales concretas que faciliten la comprensión, documentación y el aprendizaje de forma específica según el tipo de conceptos que se utilizan en programación y que además puedan ser generadas automáticamente a partir del código.

Para este propósito, es necesario conseguir un compromiso adecuado entre la abstracción asociada a los elementos visuales y su analogía semántica con el elemento representado. Además resulta también útil hacer que estas representaciones sean atractivas para el estudiante durante el aprendizaje.

2.2. Contexto taxonómico del uso de gráficos en programación

Los sistemas que usan gráficos para programar, depurar y entender la computación en general han sido motivo de investigación desde el inicio de la informática. La idea de hacer más accesible la programación a cualquier tipo de usuario ha sugerido la utilización de gráficos o lenguajes visuales para la confección de programas. De igual forma el uso de gráficos facilita la comprensión de lo que el programa hace así como su depuración cuando se enseña a programar a los estudiantes.

En este contexto muchas veces los términos utilizados para el uso de gráficos son a menudo informales o confusos. A finales de los 80 Myers [4] estableció una taxonomía más formal para los diferentes sistemas gráficos utilizados en aquel entonces y que continúa siendo válida. De esta forma establece la visualización de programas como el uso de gráficos que ilustran algún aspecto del programa o de su ejecución. En la figura 2 se muestra un ejemplo de visualización de código de forma estática.

En concreto se clasifican según se visualiza el código, los datos o el algoritmo. Tal visualización además puede ser estática o dinámica dependiendo de si existe algún tipo de animación mientras se ejecuta el programa [5]. La propuesta de este trabajo constituye una forma de visualización de código dinámica.

La justificación se fundamenta en que el sistema visual humano está optimizado para procesar información multidimensional mientras que la programación convencional, sin embargo, es representada como un texto unidimensional. En este sentido, la capacidad del código de ser entendido por un humano, es decir, su legibilidad, es discutida a continuación y evaluada a través de algunas herramientas de ejemplo. En tales ejemplos se pone de manifiesto la necesidad de exploración y explotación de nuevas metáforas conceptuales que, aplicadas a un contexto puramente computacional, sean adecuadamente flexibles para expresar, resumir, documentar y comprender mejor el código.

El videojuego de plataformas, como metáfora animada de programa, resulta especialmente adecuado para los lenguajes Turing-completos (como veremos

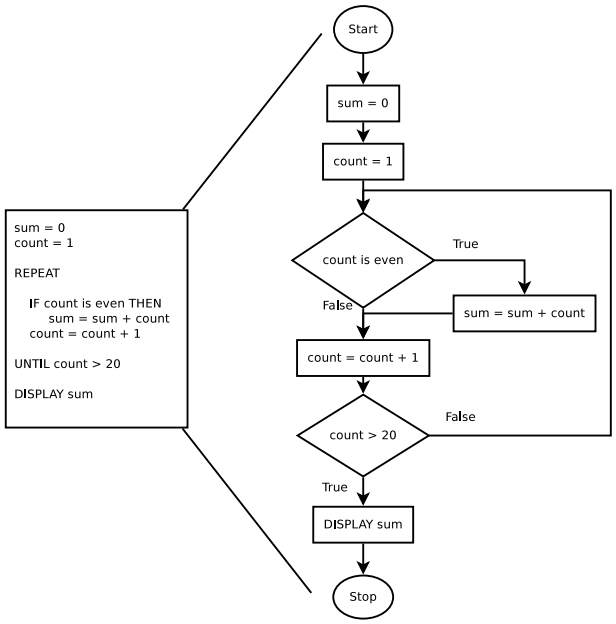

Figura 2. El diagrama de flujo visualiza el código asociado de forma estática.

posteriormente), ha surgido en el contexto de la programación y el estudiante está familiarizado con éste.

2.3. Legibilidad

A menudo se hace analogía entre el texto de un programa y el texto narrativo. La legibilidad en este sentido, como concepto, es decir, la capacidad de que algo sea leído por un humano, es una idea general muy amplia. El término "leído" implica todas las fases del procesamiento del lenguaje a fin de que el significado sea entendido por el lector. Con esta definición en mente vemos claramente diferencias sustanciales entre la legibilidad de un texto narrativo y el código de un programa.

La legibilidad del texto narrativo depende de la historia que se cuenta y de cómo el lector puede representar, mediante su imaginación, el significado natural de esta historia conforme a su experiencia. Si la historia es confusa o carece de sentido, el texto es poco legible. Sin embargo, es posible que la historia tenga pleno sentido y sea clara y que el texto siga siendo poco legible, por ejemplo, imaginemos un niño leyendo un artículo científico. En este caso la experiencia del lector es el factor predominante.

Si aplicamos esta misma idea al texto de un programa, resulta obvio que la "historia" que expresa el programa poco tiene que ver con la experiencia del lector. Su significado existe en un contexto computacional, matemático-lógico

completamente distinto, generalmente abstracto, y por tanto contrapuesto, al mundo real del lector. Para que el texto del programa sea legible por tanto sólo hay dos opciones:

- o bien forzamos el contexto semántico del programa para que se adecue a la experiencia natural del lector,
- o bien reforzamos la experiencia del lector en el contexto de la programación.

La semántica del mensaje. En el primer caso se intenta conseguir que el lector se introduzca en la programación partiendo de su experiencia natural. Su alcance, si bien importante, está limitado a la generación de programas cuya salida es multimedia y presenta alguna historia interactiva. El código usado para este fin concreto se limita a un conjunto específico de instrucciones. Incluso así, este código puede no ser legible hasta que finalmente se ejecute.

Algunas herramientas que utilizan este enfoque son:

- *Squeak Etoys*: Es un entorno de desarrollo dirigido a niños y un lenguaje de programación basado en prototipos y orientado a objetos [6]. Dirigido e inspirado por Alan Kay para promover y mejorar el constructivismo. Los principales influyentes en esta herramienta son Seymour Papert y el lenguaje Logo. La figura 3 muestra un programa de ejemplo.

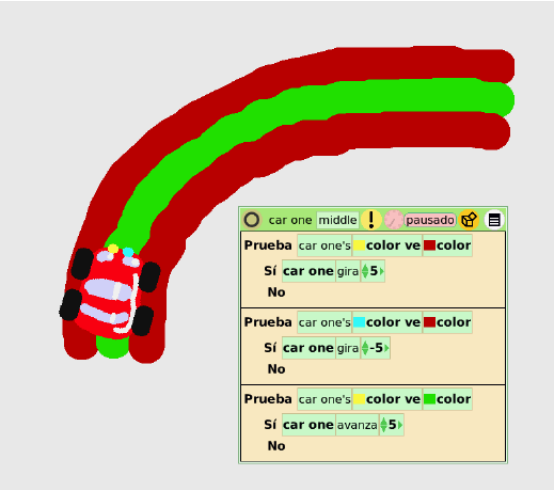

Figura 3. Un programa diseñado en Etoys para controlar un coche sigue-línea.

- *Alice*: Alice [7] es un lenguaje de programación educativo libre y abierto orientado a objetos con un entorno de desarrollo integrado (IDE). Está programado en Java. Utiliza un entorno sencillo basado en arrastrar y soltar

para crear animaciones mediante modelos 3D. Este software fue desarrollado por los investigadores de la Universidad Carnegie Mellon, entre los que destaca Randy Pausch.

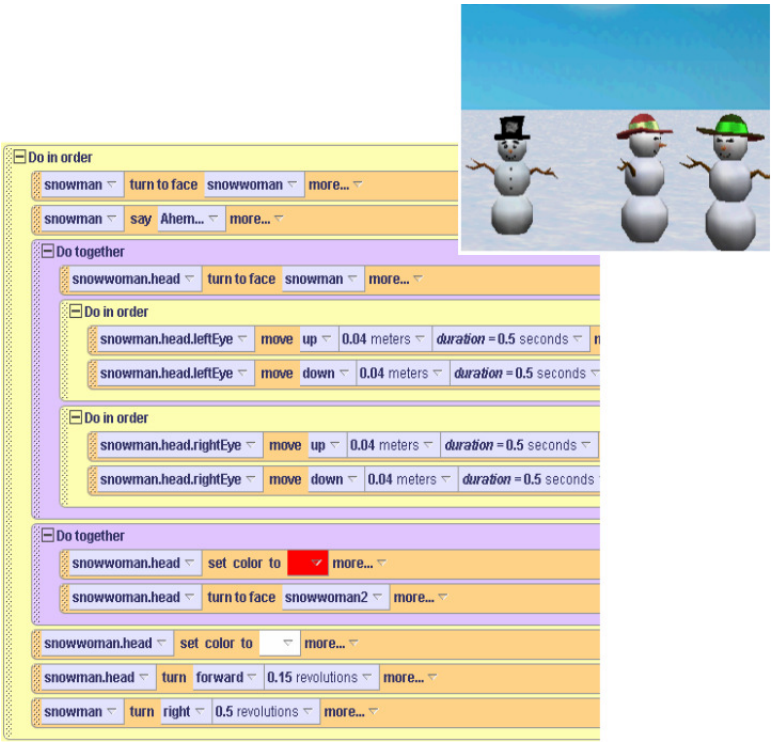

Figura 4. El mismo programa de la figura diseñado con Scratch.

- *Scratch*: Es un entorno de programación online [8] que facilita el aprendizaje autónomo. Fue desarrollado por el "the Lifelong Kindergarten group" en el Media Lab del MIT (Massachussets Institute of Tecnology) por un equipo dirigido por Mitchel Resnick y apareció por primera vez en el verano de 2007 [9]. La figura 5 muestra un programa de ejemplo. Los elementos de programación son representados como bloques de distinto color que encajan entre sí.

La experiencia del lector. El segundo caso, por otra parte, tiene un alcance mucho más amplio y por tanto, más difícil de conseguir. Es necesario trabajar multitud de conceptos como los algebraicos, lógicos y matemáticos desde edades tempranas. Además, la naturaleza abstracta de estos conceptos dificulta su

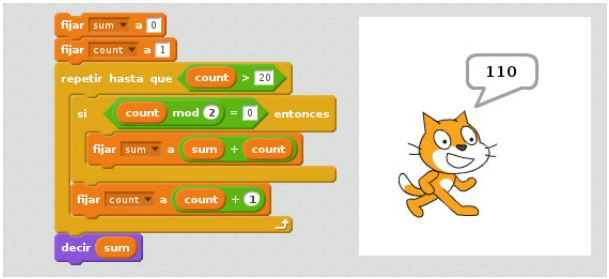

Figura 5. El mismo programa de la figura 2 diseñado con Scratch.

propia representación visual. En este sentido, algunos conceptos son más susceptibles que otros de ser visualizados, como por ejemplo, los conjuntos y sus operaciones, las funciones matemáticas, o los vectores, son más fáciles de visualizar que otros como la probabilidad, las derivadas o las diferentes partículas subatómicas. Esta visualización es importante durante el aprendizaje.

Como vemos, el concepto general de legibilidad en el ámbito de la programación no está claro. El motivo principal es claramente la confusión de no tener en cuenta que el verdadero lector efectivo de un programa no es una persona, sino una máquina.

Desde este enfoque más amplio, es posible arrojar más luz sobre la comprensión de los procesos subyacentes en el aprendizaje de la programación. De aquí la importancia de la documentación del código [10], a menudo descuidada. Es necesario abordar una automatización de generación de documentación legible entendiendo esta documentación, no sólo como un contenido añadido al programa, sino más bien como una característica intrínseca durante la generación y depuración del programa [11]. En este paradigma el programador escribe y tanto el humano como la máquina pueden leer. Puesto que la máquina y el ser humano leen de forma diferente es necesario proveer al programador de diferentes herramientas para generar paralelamente al código, distintas representaciones semánticamente representativas para otras personas. Estas herramientas son especialmente importantes durante el aprendizaje.

En este trabajo se propone una representación metafórica para visualizar el flujo de un programa de forma dinámica. La semántica utilizada para esta metáfora toma prestada la experiencia de la persona al jugar a videojuegos para hacer más intuitivo y atractivo el proceso de aprendizaje en el uso de las estructuras de control básicas de un programa escrito en un lenguaje procedural. La representación del código en forma de videojuego está motivada por la aplicación de técnicas de gamificación en el contexto educativo. Pero más allá de un uso atractivo, esta representación puede asistir también en el diseño de programas concurrentes. Pues al ser una metáfora basada en agente [2], los diferentes hilos de ejecución pueden ser representados por diferentes personajes del videojuego recorriendo un mismo escenario.

2.4. Gamificación

La gamificación consiste en el uso de mecánicas de juego o pensamiento de juego en un contexto ajeno al juego, con el fin de conseguir determinados objetivos. La gamificación aprovecha la predisposición psicológica del humano a participar en juegos. Las técnicas utilizadas consisten básicamente en tres tipos:

- *A)* Ofrecer recompensas por la realización de las tareas.
- *B)* Aprovechar la competitividad, haciendo visibles las recompensas entre los jugadores.
- *C)* Hacer más atractivas tareas ya existentes que normalmente pueden ser aburridas.

El campo de la educación presenta un gran potencial de crecimiento para la gamificación [12]. Existen multitud de ejemplos de uso de gamificación para el aprendizaje. Quizá el más llamativo es la escuela neoyorquina Quest to Learn [13], donde todo el proceso de aprendizaje está dirigido al juego.

Un ejemplo concreto de gamificación aplicado al aprendizaje de programación es Codecademy [14]. Esta plataforma online interactiva ofrece clases gratuitas de programación con diferentes lenguajes. La técnica de gamificación usada en este contexto es la de recompensar y motivar la competitividad de los estudiantes, es decir las técnicas A) y B).

En relación a la tercera técnica enumerada C), entre los ejemplos existentes podemos destacar ParticleQuest. ParticleQuest trata de enseñar a los estudiantes de física, las partículas subatómicas. La metáfora consiste en identificar las partículas subatómicas como personajes enemigos de un videojuego tipo RPG (Role-Playing Game). Las diferentes características de cada partícula identifican las habilidades y morfología de los diferentes personajes.

Figura 6. Ejemplo de gamificación: los enemigos en el juego ParticleQuest representan las diferentes partículas subatómicas.

La herramienta que se presenta en este trabajo se encuadra dentro de este último tipo de técnicas. Al visualizar el texto correspondiente al código del programa como un escenario de un videojugo, resulta familiar y divertido al usuario, y se facilita así su comprensión e interpretación.

3. Propuesta

Seymour Papert, uno de los creadores del lenguaje Logo inventó la metáfora de la pequeña-persona (*little-person*) para explicar a sus estudiantes cómo funcionaba. La metáfora consiste en que dentro del ordenador viven pequeñas personas (PPs) que son especialistas en realizar determinadas tareas, y contratan a otras PPs para realizar subtareas. Las PPs están dormidas normalmente, pero pueden ser despertadas para que realicen su tarea. Cada vez que una PP necesita ejecutar una subtarea, contrata una PP especializadad en esa tarea y se duerme. Cuando la PP contratada termina, vuelve a despertar al llamador (véase [15]).

Este trabajo propone una materialización de esta representación metafórica para visualizar los elementos del lenguaje usados en un programa. La metáfora utilizada está inspirada en las técnicas de gamificación. La idea es hacer más atractiva la tarea de codificación en programación. Para ello se utilizan videojuegos de plataformas. Quizás el videjouego de este tipo más famoso es Super Mario, para este trabajo se ha utilizado una versión libre de este juego llamada Secret Maryo Chronicles. La metáfora utilizada consiste en visualizar un método o una función, como una plataforma en el escenario del videojuego. Todo lo que hay sobre esta plataforma será el contenido del método. El uso de las estructuras de control básicas de un programa se corresponden con otras plataformas o elementos del videojuego por los que el protagonista del videojuego se mueve. Esta analogía resulta muy intuitiva puesto que relaciona la dinámica de ambos contextos. En concreto, entre los elementos básicos que pueden ser representados, se incluyen las sentencias de control típicas como el condicional, el switch, los bucles y la instrucción de retorno como se explica a continuación:

- *Condicional*: Es representado mediante una plataforma elevada. Si la condición se cumple, el personaje saltará a la plataforma, si no, pasará por debajo.
- *Switch*: Es representado mediante varias plataformas elevadas apiladas en vertical una para cada caso. El personaje saltará a la plataforma del caso que se cumpla.
- *Bucle*: Es representado mediante una tubería o similar por donde el personaje se introduce y aparece al principio del bucle.
- *Retorno*: Una puerta representa la devolución de datos del método.

Para que el flujo de programa pueda ser representado al menos se requieren estos elementos. Existen multitud de videojuegos de plataformas que cumplen con estos requisitos. En este aspecto la visualización puede ser personalizada por el usuario eligiendo los gráficos relacionados con el juego que le resulte más

atractivo. Además, otros elementos del lenguaje pueden ser representados mediante diferentes gráficos del juego. Por ejemplo, las expresiones pueden consistir en cajas que el personaje golpea para ver su contenido. Según este esquema el programa de las figuras 2 y 5 queda representado en la figura 7.

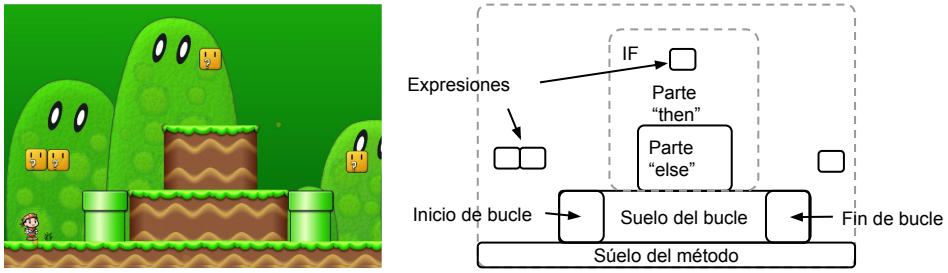

Figura 7. Visualización propuesta para el programa de ejemplo de la figura 2 y 5. El bucle está representado como la plataforma que hay entre los dos tubos, el IF dentro del bucle es otra plataforma interior y las diferentes expresiones son cajas.

Otra característica interesante para dar más capacidad de expresión a la representación visual (en el sentido opuesto al ejemplo comentado de las flechas de la sección anterior), es la posibilidad de elegir diferentes texturas para las plataformas en el contexto del juego para diferentes partes del código. Por ejemplo, supongamos que hay una parte del código más interesante o que requiere mayor atención, entonces se pueden utilizar gráficos correspondientes a escenarios más difíciles del juego como mazmorras, escenarios de lava o nieve, que sean fácilmente identificables. En la figura 8 vemos un ejemplo.

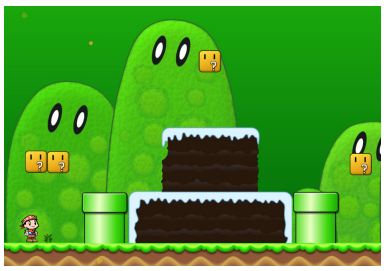

Figura 8. Visualización donde se ha resaltado todo el código del bucle utilizando plataformas con nieve.

El lenguaje para el que se ha implementado la herramienta de visualización propuesta es Java. El motivo de esta elección es el amplio uso del mismo, y la facilidad que poseen las herramientas de desarrollo para ser extendidas.

4. Implementación

Para implementar esta herramienta de visualización dinámica se ha optado por utilizar el lenjuaje Java sobre Eclipse. Eclipse es una plataforma extensible para construir IDEs. Proporciona servicios básicos para utilizar varias herramientas que ayudan en las tareas de programación. Los desarrolladores de herramientas pueden contribuir a la plataforma envolviéndolas en forma de componentes llamados plugins. El mecanismo básico es añadir nuevos elementos de procesamiento a los plugins existentes. Mediante un manifiesto XML se describe la forma en que el entorno de ejecución de Eclipse activará y manejará las instancias del plugin.

El plugin Code Reimagined consiste en una vista añadida a la perspectiva Java para mostrar la imagen generada a partir del código que se está editando.

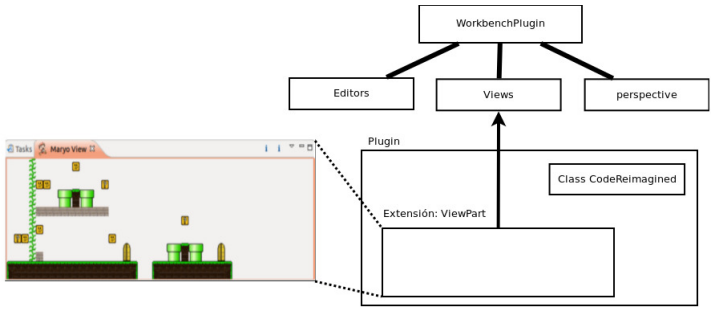

Figura 9. Esquema del plugin como una nueva vista en Eclipse.

Para analizar el código se ha usado el patrón "visitor" sobre el árbol sintáctico abstracto del código Java. Recorriendo este árbol se obtiene otro árbol cuyos nodos representan las áreas a dibujar en la vista. Estos nodos heredan de una clase abstracta y especifican cómo se pinta el elemento de programación dado y qué recuadro resulta de ello. Para obtener la imagen en la vista, basta hacer un recorrido de este árbol (sin importar el orden) y llamar al método que dibuja recursivamente los sprites de cada nodo.

Para representar la dinámica de Maryo moviéndose sobre la vista se ha implementado un *listener'* para el cursor del editor de Java, de forma que el personaje aparecerá en el área correspondiente al elemento Java que haya inmediatemente después (figura 10). La manera de hacerlo ha sido utilizar una tabla hash indexando los nodos correspondientes a los elementos Java según su offset en el archivo.

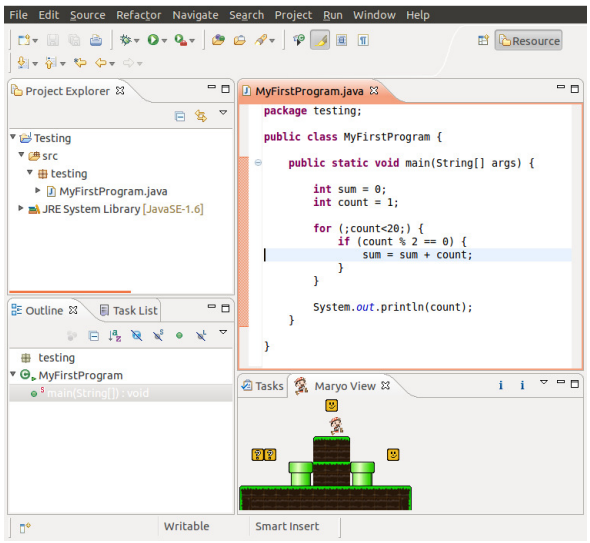

Figura 10. El plugin Code Reimagined muestra el programa de la figura 2 y 5 una vez integrado en el entorno Eclipse. Maryo aparece sobre el área dibujada correspondiente al elemento Java que hay a continuación a partir de la posición actual del cursor.

Otra funcionalidad implementada es el posicionamiento del cursor en el editor Java, al hacer doble clic sobre el elemento correspondiente del mapa.

Este plugin se presentó al VII Concurso Universitario de Software Libre [1] quedando finalista en la final local de la Universidad de Granada. El código está disponible para descarga en `https://github.com/javiplay/code-reimagined`.

5. Conclusiones

En el contexto taxonómico de la visualización de programas y partiendo del concepto de legibilidad del código, este trabajo ha identificado la necesidad de desarrollar técnicas de visualización automáticas, donde la metáfora conceptual juegue un papel básico, no sólo para el aprendizaje sino también para la documentación y la comprensión de las tareas y conceptos relacionados con la programación.

Para este tipo de visualización se ha establecido un principio básico de diseño que consiste en maximizar la precisión semántica de las representaciones minimizando la pérdida de analogía con los conceptos representados. Para la aplicación de este principio se ha comprobado la utilidad de las técnicas de gamificación, concretamente el uso de videojuegos de plataformas para representar el flujo de un programa.

[1] `www.concursosoftwarelibre.org`

La representación propuesta es independiente del lenguaje y por tanto, complementaria a cualquier software de programación y especialmente adecuado para herramientas de aprendizaje constructivista como Scratch. Una vez concretados los detalles de esta representación, se ha desarrollado una herramienta de visualización dinámica de código Java en forma de plugin para Eclipse.

Como líneas de actuación futura, además de añadir nuevas características, se propone la realización de experimentos para validar la efectividad de esta herramienta. En caso de obtener resultados positivos, se podría aplicar esta forma de visualización "gamificada", a otro tipo de diagramas relacionados con la programación orientada a objetos.

Agradecimientos

Este trabajo ha sido apoyado por el proyecto de excelencia EVORQ con referencia P08-TIC-03903 de la Junta de Andalucía, el proyecto Anyself con referencia TIN2011-28627-C04-02 del Ministerio de Innovación y Ciencia y el proyecto 83, CANUBE, concedido por el CEI-BioTIC de la Universidad de Granada.

Referencias

1. S, D.: Visualization basics. In: Software Visualization. Springer Berlin Heidelberg (2007) 15–33
2. Travers, M.D.: Programming with agents: New metaphors for thinking about computation. (1996)
3. Kumar, B.: Gamification in education-learn computer programming with fun. INTERNATIONAL JOURNAL OF COMPUTERS & DISTRIBUTED SYSTEMS **2**(1) (2012) 46–53
4. Myers, B.A.: Taxonomies of visual programming and program visualization. Journal of Visual Languages & Computing **1**(1) (1990) 97–123
5. Urquiza-Fuentes, J., Velázquez-Iturbide, J.Á.: A survey of successful evaluations of program visualization and algorithm animation systems. ACM Transactions on Computing Education (TOCE) **9**(2) (2009) 9
6. Etoys, S.: http://www.squeakland.org/ (2013)
7. Alice: http://www.alice.org/ (2013)
8. Scratch: http://scratch.mit.edu/ (2013)
9. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Communications of the ACM **52**(11) (2009) 60–67
10. Tenny, T.: Program readability: Procedures versus comments. Software Engineering, IEEE Transactions on **14**(9) (1988) 1271–1279
11. Baecker, R.: Enhancing program readability and comprehensibility with tools for program visualization. In: Software Engineering, 1988., Proceedings of the 10th International Conference on, IEEE (1988) 356–366
12. Lee, J.J., Hammer, J.: Gamification in education: What, how, why bother? Academic Exchange Quarterly **15**(2) (2011) 146
13. Salen, K., Torres, R., Wolozin, L.: Quest to learn: Developing the school for digital kids. MIT Press (2011)

14. Codecademy: http://www.codecademy.com/ (2013)
15. Harvey, B.: Computer science LOGO style. Vol. I: intermediate programming. Massachusetts Institute of Technology (1985)