Minecraft Programable: una herramienta para aprender programación en nivel medio

Programmable Minecraft: a tool to learn programming in high school

Gonzalo Zabala, Laura Pérez Cerrato, Sebastián Blanco,
Ricardo Morán y Matías Teragni
Centro de Altos Estudios en Tecnología Informática, Facultad de Tecnología Informática,
Universidad Abierta Interamericana, Argentina.
E-mail: gonzalo.zabala@uai.edu.ar; lauraperezcerrato@gmail.com;
sebastian.blanco@uai.edu.ar; ricardo.moran@uai.edu.ar; matias.teragni@uai.edu.ar

Resumen

La aparición de dispositivos inteligentes que de una forma u otra tienen un desarrollo de software interno (celulares, electrodomésticos, automóviles, entre otros), ha incrementado la necesidad de programadores. Dado que la educación formal no ha logrado cubrir esta necesidad, han surgido diversos proyectos con herramientas que permiten aprender a programar en forma sencilla y entretenida, como por ejemplo, La Hora del Código (http://code.org), que basa sus tutoriales en diferentes juegos. Dentro de esta línea se construyó una variante de Minecraft que permite en tiempo real crear y modificar el comportamiento de sus objetos utilizando Smalltalk como lenguaje.

Palabras clave: aprendizaje de programación; Minecraft; Smalltalk; Moodle.

Abstract

The rise of smart devices which in one way or another have an internal software development (mobile phones, home appliances, cars, among others), has increased the need for programmers. Since formal education has not been successful in meeting this need, several projects with tools that enable to learn programming in a simple and entertaining way have arisen, such as The Hour of Code (http://code.org), which bases its tutorials in different games. Following this trend, a variant of Minecraft, which allows creating and modifying the behavior of its objects in real-time using Smalltalk as a language, was created.

Key words: programming learning; Minecraft; Smalltalk; Moodle.

Fecha de recepción: Marzo 2016 • Aceptado: Abril 2016

ZABALA, G.; PÉREZ CERRATO, L.; BLANCOZ, S.; MORÁN, R. y TERAGNI, M. (2015). Minecraft Programable: una herramienta para aprender programación en nivel medio. *Virtualidad, Educación y Ciencia*, 12 (7), pp. 113-124.

Introducción

La tecnología atraviesa todos los aspectos de la vida humana. Muchos de los objetos de uso cotidiano poseen algún tipo de circuito integrado con poder de cómputo para cumplir con su propósito. Y para ello, necesitan algún tipo de software para su funcionamiento. Autos, televisores, celulares, microondas, cocinas, ascensores, y otros, tienen un desarrollo de programación por detrás para controlar esa computadora interna. Por este motivo surge el término "everyware", o software omnipresente (Greenfield, 2010). Sin embargo, Argentina no cuenta con la suficiente cantidad de recursos humanos para responder a este desarrollo exponencial, lo que profundiza su dependencia externa. En el año 2015, las empresas del sector rotaron más del cuarto de sus trabajadores, lo que da cuenta de la alta demanda y baja oferta de recursos calificados en el sector. En todos los tipos de perfiles técnicos, más del 20% de las empresas tuvieron dificultades en cubrir su demanda (OPPSI, 2016).

Por otro lado, el aprendizaje de lenguajes de programación y la creación de algoritmos permite desarrollar lo que se denomina pensamiento computacional. Este concepto abarca capacidades tan diversas como la abstracción, la conceptualización, la automatización de procesos, la representación mediante modelos, la división de problemas en subproblemas y otras (Wing, 2008). Es por ese motivo que en los últimos años han comenzado diversas campañas que fomentan el acercamiento a los lenguajes de programación de chicos en edad escolar. Dentro de este contexto se construyó este proyecto, buscando generar recursos didácticos que presenten una opción motivadora, y a la vez rica en contenido, para el aprendizaje de la programación.

Minecraft y los juegos sandbox

Minecraft es un videojuego de construcción donde sus jugadores pueden crear libremente estructuras utilizando diversos tipos de materiales, representados por cubos con texturas. Además, pueden acceder a distintos tipos de herramientas y recursos para enriquecer sus construcciones (Martínez, et al., 2014). Esporádicamente, y en ciertos mundos, pueden aparecer personajes peligrosos, como los zombies, pero en general no es un juego donde se corre riesgo de perder vidas o energía. Simplemente permite construir un mundo nuevo, enriquecido en red por el aporte de miles de participantes. No hay niveles, no hay misiones, no hay derrota. Sólo un universo libre donde la imaginación es el único límite.

A este tipo de juegos se los conoce como sandbox, juegos de mundos abiertos, donde no hay un camino correcto para llegar a un final (si es que hay alguno) (Squire, 2008). Son reconocidos como espacios de expresión creativa y colaborativa. Este concepto incluye a juegos de simulación como Sim City, Sim Ant, o el controvertido GTA. En mayor o menor medida estos juegos no tienen un recorrido establecido ni un final determinado: simplemente se juegan (Olson, 2010).

Teniendo en cuenta la pasión generada en los niños, y las posibilidades de creación libre que tiene Minecraft, muchos docentes comenzaron proyectos de uso en la escuela. En general, todos ellos están basados en la construcción de estructuras (Martínez, et al., 2014). Uno de los proyectos más amplios es Minecraft Edu, una versión modificada para ser usada en clase. La empresa que lo

desarrolló ofrece servicios de hosting para su uso, una biblioteca de lecciones y actividades para implementar con el juego. Una comunidad de más de 5500 docentes participan de sus foros usando Minecraft para disciplinas como Prácticas del Lenguaje, Historia y otros.

Existen otras opciones donde no hace falta una versión adaptada, pero para aprovechar al máximo su potencial, es necesario poder modificar no sólo los mapas, sino el comportamiento de los objetos presentes en el juego. Si además, estas modificaciones las pueden hacer los estudiantes, adentrándose en la lógica de comportamiento de los personajes y objetos en Minecraft, las posibilidades se extienden infinitamente.

Objetivos

El objetivo principal de este proyecto es acercar a los estudiantes al mundo de la programación de una manera atractiva y motivadora, mediante el uso de un videojuego popular y de alta inserción como Minecraft. Con este objetivo se desarrolla una nueva versión del juego que permite ver y modificar en tiempo de ejecución tanto las propiedades como el comportamiento de sus objetos.

Desarrollo del proyecto

Desde el punto de vista técnico, el proyecto consiste de dos "módulos". Uno de los módulos actúa como interfaz de programación de aplicaciones (API, del inglés: "*Application Programming Interface*"). El código de este módulo incluye un modelo simplificado del juego. Es decir, contiene un conjunto de clases y métodos que representan y permiten simular a las entidades del juego (bloques, jugadores, el mundo, etc.). Esta API está desarrollada dentro del entorno Smalltalk (Squeak).

Se decidió utilizar este entorno principalmente porque la filosofía de Squeak se basa en una idea de la programación muy diferente al resto de los lenguajes de programación. En lenguajes más tradicionales el programa que el usuario desarrolla no es más que un conjunto de archivos de texto (código fuente) que una aplicación externa (ya sea un compilador, intérprete, o una máquina virtual) puede leer, entender, traducir a código de máquina o directamente ejecutar. En Squeak, por el contrario, no existe el concepto de "programa" como entidad independiente del entorno en el que se ejecuta, no existen archivos de código fuente, ni el clásico ciclo: editar, compilar, y ejecutar. En Squeak el usuario sólo modifica un programa: el mismo Squeak. Esto se debe a que Squeak, por diseño, está escrito dentro de sí mismo y su código fuente sólo puede verse y modificarse desde dentro del mismo sistema mientras éste está en ejecución. Por esta razón, algunos autores describen a Squeak como un lenguaje de programación y como un entorno integrado de desarrollo (algunos incluso como un sistema operativo[1]). Respetando el paradigma de programación orientada a objetos, en Squeak todos los conceptos son modelados mediante objetos, incluso el código fuente del sistema mismo. Y dado que los objetos pueden ser inspeccionados, modificados y manipulados directamente (Squeak provee de herramientas gráficas para hacer esto), resulta muy sencillo modificar Squeak como desee el usuario. Es un entorno que fomenta la experimentación y el descubrimiento, al permitir no sólo ver los detalles internos de su implementación sino también modificarlos mientras el mismo está en ejecución. Y el hecho de que no requiera un proceso de compilación permite ver los efectos de las

1 https://sourceforge.net/projects/squeaknos/

modificaciones de forma inmediata. Este mismo espíritu se desea transmitir a Minecraft. Así como un programador que se pregunta cómo funcionan, por ejemplo, las estructuras de control en Squeak puede abrir un navegador de código y leer la implementación, un chico que, jugando Minecraft, se pregunta cómo funciona el comportamiento de los "creepers" podría abrir un navegador de código y fijarse. Mejor aún, podría experimentar cambiando el código del "creeper" y viendo cómo los cambios afectan su comportamiento.

El segundo módulo es una modificación del código o "mod" de Minecraft y su tarea es exponer los objetos del juego a la API, permitiendo que las instrucciones ejecutadas desde la API impacten en el estado del juego. Está implementado utilizando la plataforma Java al igual que el Minecraft original. Si bien existen múltiples interfaces de programación destinadas a facilitar la tarea de realizar modificaciones al código de Minecraft, ninguna ofrece el nivel de libertad necesario para cumplir con los objetivos del proyecto. En algunos casos, el usuario puede interactuar con el juego ejecutando instrucciones en tiempo de ejecución, pero no puede ver y modificar el código fuente del mismo. En otros, el código fuente está expuesto pero las modificaciones no pueden hacerse en tiempo ejecución, sino que se debe detener el juego, editar el código, compilar, y volver a ejecutar. Estas limitaciones resultan incompatibles con la experiencia altamente interactiva de Squeak y, por lo tanto, hacen imposible cumplir con los objetivos del proyecto.

Inicialmente se evaluaron las dos APIs más populares: Minecraft Forge y Bukkit. En ambos casos, la funcionalidad para interactuar con los objetos base de Minecraft está limitada exclusivamente a los lugares donde la interfaz lo permite. Sin embargo, modificaciones de mayor magnitud como las que el presente proyecto se propone, resultan imposibles mediante el uso de estas APIs, por lo que se decidió no emplearlas. Algunas herramientas, como LiteLoader, se descartaron debido a que sólo soportan modificaciones en el cliente de Minecraft pero no en el servidor, lo cual limita las posibilidades de interacción virtual entre los alumnos. Y otras herramientas, como Risugami's ModLoader y ModLoaderMP, se descartaron debido a que su desarrollo fue discontinuado y sólo soportan versiones antiguas de Minecraft. La herramienta que más se acerca a los objetivos del presente proyecto es ScriptCraft, un plugin para el servidor de Minecraft que permite extender el juego exponiendo una API en Javascript. Para esto, ScriptCraft utiliza la consola de comandos del juego para ejecutar código directamente en tiempo de ejecución, a la vez que permite escribir scripts más sofisticados en archivos de texto y luego ejecutarlos dentro del juego. Tiene una funcionalidad muy amplia y la documentación es excelente. Sin embargo, no permite ver ni editar el código mismo de Minecraft (ni del plugin) en tiempo de ejecución. La API sólo permite interactuar con los objetos del juego escribiendo scripts que modifiquen sus propiedades, pero no ofrece ninguna herramienta de exploración del código ni de depuración, lo cual limita las posibilidades de aprendizaje que este proyecto se propone explorar. Finalmente, otro proyecto a destacar es la interfaz entre Squeak y Minecraft Pi desarrollada por Bert Freudenberg[2]. Este proyecto, de similares características, sólo funciona con la versión de Minecraft para Raspberry Pi, en lugar del Minecraft convencional. Y a pesar de que permite controlar Minecraft desde Squeak, está limitado a la interfaz de programación integrada en Minecraft Pi.

2 http://croquetweak.blogspot.com.ar/2013/02/smalltalk-bindings-for-minecraft-pi.html

Dado que Minecraft es un producto cerrado, inicialmente se evaluó trabajar sobre un clon de código abierto (como por ejemplo Craft[3]), pero esta alternativa fue descartada ya que no se halló ningún juego semejante en un estado de desarrollo estable y con la misma funcionalidad. También se consideró el desarrollo de un clon de Minecraft propio, pero esta opción no fue seleccionada por el tiempo de desarrollo que implicaría para el equipo, y en su lugar se decidió trabajar con el juego original. Para ello se utilizó un conjunto de herramientas conocido como "Mod Coder's Pack" (MCP) que, mediante una herramienta llamada Fernflower, permite decompilar y deofuscar el código original.

Temporalmente, se decidió mantener independientes ambos módulos y utilizar un mecanismo de comunicación entre procesos para implementar la interacción entre ambos. Esto permite comenzar el diseño y desarrollo de la API al mismo tiempo que se desarrolla una integración más profunda entre Minecraft y Squeak. Pero tiene como desventaja que la ineficiencia en la comunicación limita considerablemente la funcionalidad que se puede implementar en esta primera etapa.

Las modificaciones que se hicieron al código del Minecraft consisten principalmente en la implementación de una capa de comunicación que permite a la API conectarse y enviarle instrucciones para modificar el estado del juego. Estas instrucciones fueron modeladas utilizando el patrón de diseño Command (Gamma, et al., 1995) con el objetivo de minimizar y aislar las modificaciones al código original de Minecraft, facilitando además la distribución del código del proyecto y su adaptabilidad a diferentes versiones del juego. Entre las instrucciones que fueron modeladas se encuentran las acciones básicas que puede llevar a cabo un jugador, como colocar y remover bloques, consultar la posición del jugador o el tipo de algún bloque, etc.

Las instrucciones ejecutadas utilizando la API son comunicadas al juego y, al ser ejecutadas, éste "responde" siempre con algún objeto correspondiente al mensaje que acaba de recibirse, respetando la filosofía de Smalltalk, en la cual todos los mensajes tienen siempre una respuesta (Black, et al., 2007).

Utilizando una interfaz de programación como la propuesta es posible, entre muchas otras cosas, la generación de estructuras complejas de construir manualmente mediante la adaptación al modelo de algoritmos conocidos. A continuación, se desarrollará un ejemplo paso a paso de cómo utilizar la API para construir una estructura ejecutando unas pocas instrucciones de código.

La acción más sencilla que se puede realizar en Minecraft es ubicar un bloque en algún punto en el espacio. Utilizando la API propuesta se puede crear un bloque mediante el siguiente método, donde se especifica el tipo de bloque y su ubicación en el espacio:

```
MinecraftExamples>>block: type at: point3D
      world addBlock: (MinecraftBlock
                  ofKind: (MinecraftBlock perform: type)
                  at: point3D)
```

3 https://github.com/fogleman/Craft

Se puede utilizar este método para, por ejemplo, ubicar un bloque en una posición relativa al jugador:

```
examples block: #stone at: player position + 10
```

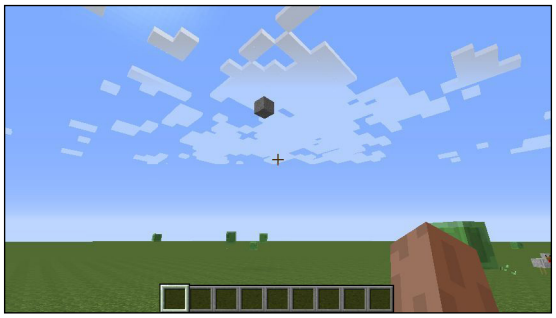

Figura 1. Creación de un bloque.

Extendiendo éste método en dos dimensiones resulta sencillo construir paredes, como puede observarse en el siguiente código:

```
MinecraftExamples>>
wall: type at: point width: width height: height
      point x to: point x + width do: [:x |
      point y to: point y + height do: [:y |
            self block: type at: x @ y @ point z]]
```

El usuario simplemente debe especificar el tipo de bloque que se usará para la construcción, el punto de partida, el ancho y el alto de la pared. Por ejemplo:

```
examples wall: #stonebrick
      at: player position - (0 @ 0 @ 10)
      width: 10

      height: 10.
```

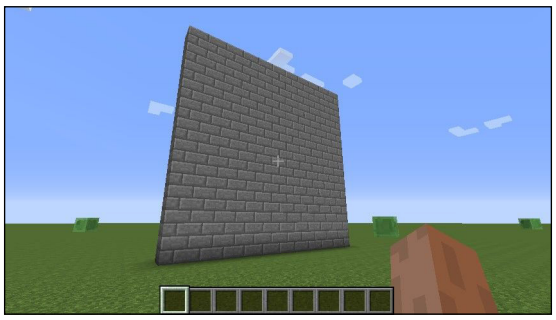

Figura 2. Una pared de 10 por 10 ladrillos en una posición relativa al jugador.

Una generalización de este método en las 3 dimensiones permite crear un conjunto de bloques especificando sólo 2 puntos:

```
MinecraftExamples>>
blocks: type from: point1 to: point2
      point1 x to: point2 x do: [:x |
      point1 y to: point2 y do: [:y |
      point1 z to: point2 z do: [:z |
            self block: type at: x @ y @ z]]]
```

Este método permite, no sólo construir paredes, sino también cubos y otros ortoedros:

```
examples
      blocks: #cobblestone
      from: player position + (-5 @ 0 @ -20)
      to: player position + (5 @ 10 @ -10).
```

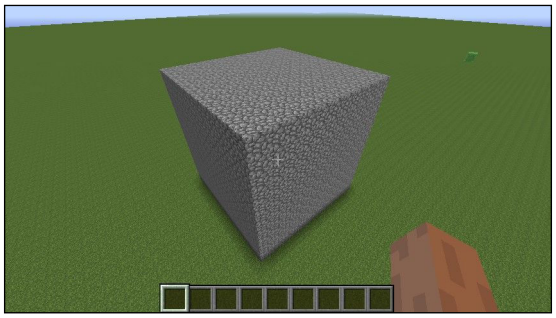

Figura 3. Un método para crear cubos de ladrillos.

Si agregamos algunos detalles decorativos, se puede construir fácilmente una torre de vigilancia:

```
MinecraftExamples>>towerAt: point height: height
      self blocks: #stonebrick
            from: point - (1 @ 0 @ 1)
            to: point + (1 @ (height - 1) @ 1).
      self block: #stonebrick
            at: point + (-1 @ height @ -1).
      self block: #stonebrick
```

```
            at: point + (-1 @ height @ 1).
      self block: #stonebrick
            at: point + (1 @ height @ -1).
      self block: #stonebrick
            at: point + (1 @ height @ 1).
      self block: #torch
            at: point + (-2 @ (height - 2) @ 0).
      self block: #torch
            at: point + (2 @ (height - 2) @ 0).
      self block: #torch
            at: point + (0 @ (height - 2) @ -2).
      self block: #torch
            at: point + (0 @ (height - 2) @ 2)
```

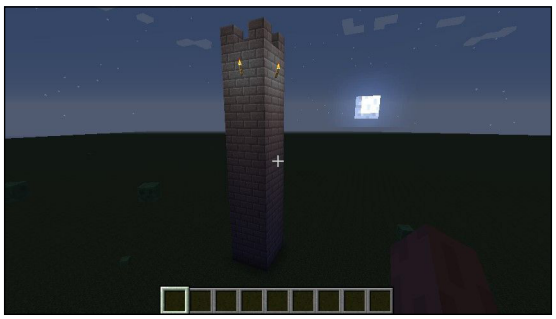

Figura 4. Resultado de la construcción de la torre de ladrillos.

Y, combinando los distintos métodos, se puede lograr una muralla:

```
MinecraftExamples>>fort: point1 to: point2 height: height
      | wallHeight |
      self towerAt: point1 height: height.
      self towerAt: point2 height: height.
      self towerAt: (point1 x @ point1 y @ point2 z)
            height: height.
      self towerAt: (point2 x @ point1 y @ point1 z)
            height: height.
      wallHeight := height - 4.
      self blocks: #stonebrick
```

```
        from: point1
        to: (point1 x @
              (point1 y + wallHeight) @
             point2 z).
self blocks: #stonebrick
        from: point1
        to: (point2 x @
              (point1 y +wallHeight) @
             point1 z).
self blocks: #stonebrick
        from: (point1 x @ point1 y @ point2 z)
        to: point2 + (0 @ wallHeight @ 0).
self blocks: #stonebrick
        from: (point2 x @ point1 y @ point1 z)
        to: point2 + (0 @ wallHeight @ 0).
```

Figura 5. Construcción de un castillo utilizando los métodos anteriores.

Este pequeño ejemplo muestra cómo, mediante el uso de unas pocas instrucciones que forman la base de toda la API de Minecraft, pueden generarse estructuras que llevarían mucho tiempo construir a mano. Por una cuestión de espacio, sólo se incluyó el código para construir una estructura relativamente simple como es una muralla, pero otras estructuras más complejas también son posibles. Por ejemplo, fractales como el triángulo de Sierpinski (Figura 6) o la curva del dragón (Figura 7).

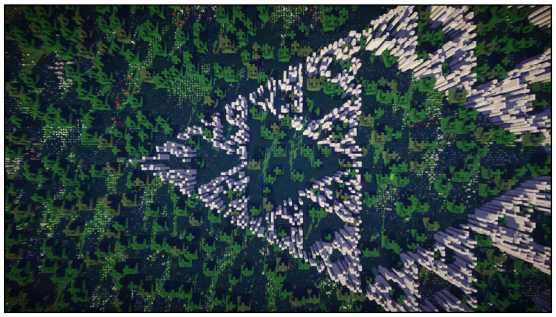

Figura 6. Triángulo de Sierpinski.

Figura 7. La curva del dragón.

Todos estos ejemplos vienen preinstalados en la API para que los jugadores puedan tomarlos como referencia y desarrollar sus propias estructuras. Por supuesto, todo el código que compone a la API es de código abierto. En el caso del código del juego, si bien el Minecraft original fue decompilado y modificado el objetivo no es exponer al jugador a ese código sino en su lugar portar la mayor parte al módulo en Squeak. De esta forma, el usuario tendrá una única interfaz de programación homogénea que ver, modificar, y extender.

Trabajo futuro

El proyecto, en su estado actual, aun no cumplió con los objetivos planificados. Por el momento, sólo permite agregar y quitar bloques del escenario. Como trabajo futuro se espera poder realizar modificaciones más importantes al juego. Por ejemplo, cambiar el comportamiento de un tipo de bloque o desactivar la inteligencia artificial de las entidades y reemplazarla por una propia. En este sentido, la interacción entre los módulos debe reemplazarse por un mecanismo de comunicación más eficiente. Existen diferentes alternativas a evaluar:

1. Alterar el código de Minecraft de forma que la ejecución de ciertas partes del código se convierta en la ejecución de scripts análogos de lenguajes dinámicos desarrollados para la plataforma Java (por ejemplo, Groovy). La principal ventaja de esta estrategia es la sencillez de su implementación (la interacción entre lenguajes que utilizan la máquina virtual de Java es rápida en su implementación y, en el caso de Groovy, la sintaxis de este lenguaje es tan similar a la de Java que la traducción del código de Minecraft a Groovy podría resultar innecesaria) pero implica una pérdida de performance que podría resultar inaceptable para esta aplicación.

2. Implementar dentro del código de Minecraft la carga dinámica de clases o hot swapping: de forma análoga, llevar a cabo la compilación y carga dinámica de clases no implicaría una pérdida de performance tan significativa como la opción anterior, pero se perdería la simplicidad de aquella. Cabe añadir que existen en la actualidad herramientas (comerciales y no comerciales) que facilitarían su implementación.

3. Incorporar al código de Minecraft una máquina virtual de Smalltalk desarrollada para la plataforma Java: contando con una máquina virtual de Smalltalk lo suficientemente optimizada, esta alternativa permitiría mantener la mayor parte del código de la API y extenderla de forma que puedan cumplirse todos los objetivos propuestos. Sin embargo, su implementación resulta altamente compleja y aunque existen varias implementaciones de Smalltalk en Java (por ejemplo,

JSqueak[4], Potato[5], o Redline[6]) todavía no se evaluó si cumplen con los requisitos necesarios para el proyecto.

Ninguna de estas alternativas facilita la tarea de incorporar la funcionalidad del proyecto al modo multijugador, coartando el aspecto colaborativo. Estrategias similares podrían ser aplicadas, modificándose el código del servidor del juego, con ese fin.

Conclusión

Se describió la implementación de una API para un videojuego popular como Minecraft que permite utilizar la programación como medio para interactuar con el juego. En esta implementación el uso de programación es completamente opcional, el juego funciona exactamente igual sin ella. Sin embargo, escribiendo código se accede a facilidades imposibles de obtener jugando de forma manual. Por ejemplo, se pueden construir estructuras complejas en cuestión de segundos o, en una versión futura, modificar el comportamiento de las entidades del juego. La existencia de una API con las características ya mencionadas puede llegar a fomentar el aprendizaje de programación incluso aunque no se usara en el aula como herramienta didáctica. Este enfoque pone énfasis en el uso de la programación como medio y no como fin. No se intenta enseñar programación. En su lugar se intenta mostrar el valor de la programación, en este caso mediante un videojuego ya popular entre los estudiantes. Si se puede usar una herramienta que ayude a los chicos a apreciar la utilidad de la programación en un contexto que les interesa, esta herramienta tiene potencial para motivar a cada vez más estudiantes a ingresar a carreras vinculadas con la programación.

Referencias Bibliográficas

BLACK, A.; DUCASSE, S.; NIERSTRASZ, O. y POLLET, D. (2007). *Squeak by Example. Square Bracket Associates*. En línea: http://SqueakByExample.org/index.html [10/02/2016]

FUNDACIÓN SADOSKY. (2013). *CC-2016 Una propuesta para refundar la enseñanza de la computación de las escuelas argentinas*. En línea: http://www.fundacionsadosky.org.ar/wp-content/uploads/2014/06/cc-2016.pdf [10/02/2016]

GAMMA, E.; HELM, R.; JOHNSON, R.; y VLISSIDES, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading Mass: Addison-Wesley.

GREENFIELD, A. (2010). *Everyware: The dawning age of ubiquitous computing*. New Riders.

MARTÍNEZ, F.; DEL CERRO, F.; y MORALES, G. (2014). El uso de Minecraft como herramienta de aprendizaje en la Educación Secundaria Obligatoria. Ponencia presentada en: *III Congreso de buenas prácticas en la atención a la diversidad*. Murcia: Consejería de Educación, Cultura y Universidades.

OBSERVATORIO PERMANENTE DE LA INDUSTRIA DEL SOFTWARE Y SERVICIOS INFORMÁTICOS (2016). *Reporte anual sobre el Sector de Software y Servicios Informáticos de la República Argentina - Año 2015*. Cámara de Empresas de Software y Servicios Informáticos de la Argentina.

4 http://www.esug.org/data/Smalltalk/Squeak/JSqueak/local.html
5 https://sourceforge.net/projects/potatovm/
6 http://www.redline.st/

OLSON, C. K. (2010). Children's motivations for video game play in the context of normal development. *Review of General Psychology*, 14 (2), pp 180-187.

SQUIRE, K. (2008). Open-ended video games: A model for developing learning for the interactive age, en Salen K. (ed.), *The ecology of games: Connecting youth, games, and learning*. Cambridge: MIT Press, pp. 167-198.

WING, J. M. (2008). *Computational thinking and thinking about computing*. Philosophical Transactions of the Royal Society A, 366 pp. 3717-3725.