Desarrollo de Juegos como Estrategia Didáctica en la Enseñanza de la Programación

Frittelli, V. – Tartabini, M. – Teicher, R. – Steffolani, F. – Serrano, D. – Fernández, J. – Bett, G. – Strub, A.
Universidad Tecnológica Nacional, Facultad Regional Córdoba

Abstract
La enseñanza de la programación de computadoras constituye un desafío exigente en cuanto a creatividad para el diseño de materiales de estudio potencialmente significativos (notas de clase, ejemplos, casos de aplicación, ejercicios, etc.) que despierten el interés y la curiosidad en los estudiantes. Esto suele ser complejo de realizar, sobre todo cuando se trata de estudiantes ingresantes y con ninguna o muy poca experiencia previa en programación, ya que en estos casos esa falta de experiencia previa hace difícil encontrar contenidos y prácticas familiares para los alumnos, que favorezcan el anclaje de nuevos elementos en sus redes conceptuales. En este trabajo, se analizan y exponen experiencias basadas en el diseño y desarrollo de diversos juegos para computadoras, como estrategia para la enseñanza de la programación. El planteo didáctico basado en juegos ofrece numerosas ventajas desde la perspectiva del aprendizaje significativo, ya que muchos de los juegos empleados son muy conocidos por los estudiantes, se basan en reglas y conceptos muy simples y cotidianos y permiten el modelado del dominio del problema con relativa sencillez.

Palabras Clave
Enseñanza de la Programación. Aprendizaje Significativo. Diseño y Desarrollo de Juegos en Programación. Experiencias con Juegos Típicos.

Introducción y Marco Teórico

La enseñanza de la programación de computadoras ha sido desde sus inicios un campo desafiante ya que implica no solamente facilitar a los alumnos el aprendizaje de uno o varios lenguajes de programación, sino fundamentalmente guiar a esos alumnos en la tarea de aprender a resolver problemas, plantear algoritmos que reflejen esas soluciones y analizar la eficiencia de esas soluciones en forma comparativa. Y el desafío es especialmente complejo cuando se trata de cursos de introducción a la programación, en los que se cuenta con alumnos que normalmente no tienen ninguna experiencia previa (o la tienen muy básica) en programación de computadoras y proceden además de diversos ámbitos académicos y/o técnicos (colegios con orientaciones diferentes, academias o institutos de formación, etc.) Para agregar complejidad al problema, existe también en muchos ámbitos universitarios una fuerte discusión sobre qué paradigma de programación debería usarse en estos cursos introductivos: por caso, la Programación Estructurada, la Programación Orientada a Objetos (POO) o la Programación Funcional (por citar sólo los más nombrados). Un curso introductivo de programación abarca un cierto conjunto de objetivos y contenidos mínimos que en general se consideran ineludibles. Pero también hay consenso general en cuanto a que esos objetivos y contenidos mínimos pueden ser tratados y trabajados desde distintos enfoques: se puede hacer un tratamiento completamente basado en Programación Estructurada (que es la estrategia más clásica), se puede canalizar una línea de acción progresiva que comience con un tratamiento elemental y estructurado, y luego ir introduciendo de a poco el paradigma de la POO, o se puede intentar comenzar directamente con la POO, tratando en forma transversal cada tema específico que pudiera requerirse a medida que se avanza. No hay un acuerdo general respecto a cuál de estos tres enfoques es el más efectivo. Se han realizado (y se siguen realizando) numerosas experiencias en una u otra dirección, arribando a diversas conclusiones (Malinowski, 1999). Como sea, el hecho es que diversas instituciones superiores

universitarias o no universitarias asumen su postura al respecto y hacen grandes esfuerzos para guiar a sus estudiantes en el camino del aprendizaje de la programación. Sea cual fuese el paradigma al que se adhiere en los cursos iniciales, la situación de base suele ser la misma: muchos de los estudiantes inscriptos en cursos de introducción a la programación manifiestan una fuerte dependencia respecto de sus profesores y de los medios externos que de alguna manera favorezcan su aprendizaje. En términos de psicología educativa, se dice que esos alumnos tienen un *foco de control externo* (Ausubel, Novak, & Hanesian, 2009) y requieren de explicaciones muy precisas en cuanto a los planteos sugeridos para resolver un problema y una detallada estructuración de clases para ayudarlos en su avance. En general, a medida que los alumnos progresan en su carrera tienden a ser más independientes de sus profesores y medios externos, para ser más autosuficientes. En este último caso, pasan a un *foco de control interno* (Ausubel, Novak, & Hanesian, 2009): los estudiantes demuestran ser más permeables a resolver casos prácticos y problemas por sí mismos, aún sin mediar una explicación previa de cómo hacerlo, o recibiendo sólo una somera introducción a la técnica resolutiva sugerida.

Aun dando por descontado ese avance, queda pendiente la cuestión de las estrategias a seguir en cursos introductivos. El plan de trabajo para un curso de introducción a la programación debería contemplar los conocimientos que el alumno haya adquirido en sus experiencias previas a su participación en el curso, para favorecer el *aprendizaje significativo* (Ausubel, Novak, & Hanesian, 2009) y lograr que efectivamente progrese: si los nuevos conceptos a estudiar y prácticas a desarrollar tienen relación efectiva con lo que el estudiante ya conoce de antemano, se da el *aprendizaje significativo*. Si no hay relación entre lo nuevo y lo que estudiante ya conoce, entonces el aprendizaje no es significativo, sino *memorístico* y *repetitivo*.

Lo anterior implica también que el alumno debe estar convenientemente motivado para relacionar lo nuevo con lo que sabe y producir el *anclaje significativo*. Un docente hábil y preparado debe saber pensar, encontrar, diseñar y/o desarrollar materiales que logren motivar al estudiante y comprender que en la mayoría de los casos el condicionante de las calificaciones no es suficiente: debe también buscar la *funcionalidad* de los conocimientos que se intenta transmitir en forma significativa, lo cual implica plantear problemas, casos, conceptos e incluso evaluaciones, que puedan ser efectivamente utilizados por los alumnos en la práctica real, o ser deducidos de ella. Cuando un problema, una evaluación o un caso de análisis se funcionaliza y se plantea en esta forma, hablamos de una *prueba* o *modelo situacional* (Ahumada, 1990). El diseño y planteo de materiales de estudio y casos prácticos con potencial significativo y correctamente funcionalizados, favorecen un componente básico del aprendizaje significativo que la *memorización comprensiva* (Ausubel, Novak, & Hanesian, 2009). La memoria es el punto de partida para nuevos aprendizajes, y no solo el recuerdo (a veces efímero) de lo que se aprendió. Si se impone la consigna de estudiar conceptos no significativos ni funcionalizados, sólo se logrará que el alumno memorice en forma mecánica, repetitiva y sin sentido.

Este trabajo pretende mostrar y analizar la forma en que el diseño y desarrollo de juegos como parte del plan de trabajo de un curso de programación introductiva puede constituir una estrategia didáctica valiosa, en el contexto pedagógico que se expuso en los párrafos anteriores a modo de marco teórico.

Elementos del Trabajo y Metodología

Planteada la necesidad de diseñar estrategias didácticas que favorezcan el aprendizaje significativo, los docentes deberían esforzarse por diseñar y funcionalizar materiales, ejemplos, prácticas y aplicaciones que se basen en los conocimientos y experiencias previas de los alumnos. El punto es que no siempre es sencillo y obvio dar con un cuerpo de conocimientos y prácticas que sean efectivamente comunes al conjunto de estudiantes que se inscriben para un curso de introducción a la programación. En el nivel superior universitario o no universitario existen ciertas exigencias de ingreso a cada carrera que ayudan a garantizar que los estudiantes efectivamente cuentan con ciertos conocimientos previos, pero el hecho es que al comenzar el desarrollo del curso esos conocimientos previos suelen aparecer dispersos o poco profundizados. Es notable además que en el grupo de alumnos algunos dominan con mayor precisión que otros ciertos conceptos previos. Surge entonces el problema del docente: ¿en qué clase de situaciones problemáticas pueden basarse los contenidos del curso, de forma que esas situaciones referencien a la red de conceptos y experiencias que los alumnos ya poseen?

Uno de los primeros contextos que se suelen analizar en la búsqueda de esas situaciones es el campo de la matemática y la geometría. Los ejemplos matemáticos básicos y especialmente los ejemplos de geometría plana elemental cubren bien estos requisitos ya que normalmente los alumnos han adquirido suficientes conocimientos básicos previos en esas disciplinas como para que sean admitidos como potencialmente significativos. Desde el punto de vista de la enseñanza de la programación, el uso de marcos de trabajo basados en ejemplos de matemática y geometría ofrece algunas ventajas y características destacables:

- *Reglas claras*: por basarse en fórmulas preestablecidas, el dominio es conocido por los alumnos y es poco discutible. En el cálculo de la superficie de una figura plana, por ejemplo, se necesita conocer ciertos datos de la figura y las fórmulas asociadas al cálculo del área para generar el resultado requerido.
- *Alcance limitado*: se pueden identificar claramente los datos necesarios para caracterizar al elemento estudiado (un polinomio, una figura geométrica, etc.) y las necesidades de cálculo requeridas. Incluso más: las fórmulas facilitan la diferenciación entre variable y constante, así como la diferencia entre cálculo y resultado.
- *Resultados interpretables*: con mínimos cálculos se puede determinar si el proceso proporciona o no el resultado correcto.

Desde la perspectiva de la POO (si ese fuese el paradigma empleado), los ejemplos básicos de matemática y geometría son especialmente útiles por diversas razones:

- Se pueden identificar fácilmente las variables de una fórmula como atributos de la figura o elemento bajo estudio.
- La propia fórmula define la responsabilidad del cálculo.
- Se puede sugerir al alumno la identificación de otras responsabilidades simples.
- Se pueden crear e involucrar varias instancias, tanto en problemas simples como en el caso de una composición. Por ejemplo, una escarapela puede ser representada como un grupo de tres círculos concéntricos y a partir de ellos calcular las superficies de las áreas de distintos colores.

Sin embargo, puede notarse que la desventaja de este tipo de ejemplos matemáticos o geométricos elementales se da en la posible falta de amplitud conceptual, lo que puede contribuir a la

pérdida de motivación por parte del estudiante. Si bien se pueden incorporar herramientas gráficas para compensar la falta de amplitud conceptual, el objetivo del programa (entendido como proceso para obtener los resultados requeridos) no representa un gran desafío para el alumno ni lo impulsa a ampliar el alcance o pensar en otros ejemplos. Por supuesto, el horizonte de ejemplos y aplicaciones de matemática y geometría para la programación no se limita a casos básicos y elementales y abarca áreas complejas e incluso abiertas a la investigación, pero la incorporación de esos elementos escaparía al planteo de un curso de programación introductorio, ya que muchos de esos problemas requieren el uso de estructuras de datos y algoritmos (Sedgewick, 1995) que en un curso inicial estarían fuera de contexto.

Lo anterior indica que las situaciones problemáticas y con potencial significativo tomadas desde la matemática y la geometría son valiosas y útiles, pero por si mismas no alcanzarían para cubrir los requisitos de un curso introductivo de programación. Y en ese sentido, el campo del diseño y desarrollo de juegos (aunque inicialmente sean básicos y elementales) ofrece incluso más ventajas y características deseables para el aprendizaje significativo que las que aportan la matemática y la geometría (aunque estas últimas pueden llegar a usarse como parte del diseño del juego propuesto):

- ✓ Desde el punto de vista de la potencialidad significativa, muchos juegos de azar son *conocidos por los estudiantes* desde la infancia, lo que permite el diseño de intervenciones didácticas que hagan referencia directa y simple a las redes de conceptos y experiencias previas de esos alumnos.
- ✓ Ofrecen muy variados y conocidos *grados de complejidad*, con lo cual se puede lograr que se mantenga un elevado nivel de motivación por parte de los estudiantes para el diseño de cada módulo, etapa o nivel del juego. Si bien el equipo docente debe seleccionar juegos que puedan ser desarrollados en un tiempo acorde al que se dispone para el curso, siempre es posible plantear un esquema que permita el cumplimiento de ciertos alcances en cada clase, sin dejar cabos sueltos, y dejar abierto el desafío de introducir mejoras y ampliaciones.
- ✓ Aun cuando los juegos forman parte de la experiencia previa de los estudiantes y ayudan al anclaje de nuevos conceptos, desde el punto de vista de un curso de programación de computadoras ofrecen un *campo novedoso* en cuanto al tipo de prácticas, modelos y ejercicios a desarrollar, contrariamente a las típicas situaciones de carácter administrativo, en las que un problema suele plantearse simplemente como una secuencia de procesamiento de datos que está enunciada en el propio problema.
- ✓ El diseño de un juego incluye el modelado y programación de numerosas reglas que serán más complejas mientras más sofisticado sea el juego, pero al igual que en el contexto matemático o geométrico, esas *reglas también son claras* de identificar, ya que sigue tratándose de situaciones que los estudiantes conocen y son familiares a su entorno. Además, los resultados del programa que modela el juego son también interpretables con facilidad por parte de los estudiantes.
- ✓ Si el paradigma adoptado por los docentes responsables del curso fuese el de la POO, el diseño y desarrollo de un juego implica la posibilidad de crear un modelo de clases de cierta complejidad y crear numerosas instancias que deben luego interactuar entre ellas, ayudando así a incorporar el concepto de colaboración. Por otra parte, el modelado del dominio del problema permite un trabajo detallado de identificación de atributos y responsabilidades que ayuda a su vez a fijar en forma natural el

concepto de encapsulamiento y otras características básicas de la POO.

Resultados

A lo largo de los últimos diez años, los autores de este trabajo han llevado a cabo numerosas experiencias basadas en el desarrollo de juegos, en cursos universitarios con diferentes niveles de profundidad (por tratarse, por ejemplo, de asignaturas de programación de distintos años en carreras de orientación informática) y con diversos perfiles de conocimientos y experiencias previas de los alumnos inscriptos. Asimismo, en los cursos citados se emplearon distintos lenguajes de programación (entre ellos C, C++ y Java) e incluso distintos paradigmas de programación (programación estructurada en algunos cursos introductorios de primero y segundo año, y programación orientada a objetos en otros cursos de primero, segundo y tercer año). A modo de resumen clasificatorio breve, se exponen y analizan a continuación las experiencias más relevantes junto con una descripción básica del juego utilizado en cada una:

a.) *Juego del Pac-Man*: Con lenguaje C y elementos de C++, usando la librería BGI para soporte de gráficos del producto Borland C++ 3.1 y paradigma de programación estructurada. El desarrollo del *Juego del Pac-Man*[1] permitió introducir a alumnos de primer año (segundo cuatrimestre) en técnicas de gestión avanzada de gráficos, simulación de movimiento, almacenamiento externo de elementos gráficos en base a formatos definidos por el programador, y programación del motor de inteligencia del juego para controlar el comportamiento autónomo de algunos personajes. Elementos destacados: el juego es visualmente atractivo y muy dinámico, requiriendo una permanente actualización del gráfico en pantalla. Combina personajes que son controlados por un jugador humano (y sólo requieren que el programador controle la validez de los movimientos solicitados y luego renderizar la imagen) con personajes que son controlados por el programa (y en ese caso el programador debe programar las reglas de comportamiento). Además, para programar este juego se requiere poner en práctica efectiva casi la totalidad de los temas y conceptos abarcados por la asignatura, durante un período de alrededor de dos meses.

b.) *Juegos de Cartas (o de Barajas)*: Con lenguaje Java y paradigma de programación orientada a objetos. El desarrollo del juego permitió introducir a alumnos de primer año (primer cuatrimestre) en técnicas de gestión elemental de gráficos y actualización de contenidos de pantalla y programación del motor de inteligencia del juego para controlar el comportamiento de los jugadores virtuales. Como el manejo de gráficos en Java requiere de algunos conocimientos previos en cuanto a gestión de ventanas y eventos que los alumnos no manejan en el momento de plantear la experiencia, los profesores desarrollaron un marco de trabajo con clases diseñadas para facilitar el despliegue de gráficos (Frittelli, Teicher, Tartabini, Fernández, & Bett, 2011) y para controlar algunos elementos complejos del juego (como la propia baraja) y los estudiantes utilizaron y ampliaron ese marco de trabajo para sus propios desarrollos. Elementos destacados: con el enfoque antes indicado, es relativamente simple proponer el planteo de juegos de cartas diferentes, eventualmente ampliando las prestaciones del marco de trabajo provisto por los profesores. Estos juegos son visualmente atractivos pero a la vez adecuadamente simples de controlar en cuanto al despliegue de gráficos. Si los juegos planteados son bien elegidos,

[1] URL de consulta: http://en.wikipedia.org/wiki/Pac-Man

entonces las reglas de control a programar son simples. Se puede combinar jugadores humanos con jugadores virtuales. Desde la perspectiva de la POO usada como paradigma en el curso, el desarrollo del juego permite poner en práctica elementos básicos como el encapsulamiento, el ocultamiento, la instanciación y colaboración entre objetos, durante un período de alrededor de un mes.

c.) *Juego del Tic-tac-toe (o Ta-te-ti)*: Con lenguaje C++, usando la librería MSDN del producto Microsoft Visual Studio y paradigma de programación orientada a objetos. El desarrollo del conocido *Juego del Tic-tac-toe*[2] permitió introducir a alumnos de segundo año (segundo cuatrimestre) en técnicas de diseño de interfaces visuales de usuario, gestión de gráficos y control de eventos del mouse, y programación del motor de inteligencia del juego aplicando estructuras de datos y técnicas muy específicas como los *árboles de juegos* (Langsam, Augenstein, & Tenenbaum, 1997), la *recursividad* y el *procesamiento en backtraking* o la *poda alfa-beta* (Weiss, 2000). Elementos destacados: el juego es visualmente simple, permitiendo que el programador pueda concentrarse más en refinar el motor de inteligencia. Su sencillez conceptual permite aplicar y probar estructuras de datos y técnicas específicas para control de juegos de tablero con oponentes, sin que la complejidad del juego abrume al programador. Se puede ajustar con facilidad a diferentes niveles de respuesta inteligente por parte de la computadora. El hecho de poder programar el motor de inteligencia del juego cambiando y combinando distintas técnicas y estructuras de datos de soporte, lo vuelven muy útil para operaciones de testing (incluso en diferentes asignaturas).

d.) *Simon Says*: Con lenguaje Java y paradigma de programación orientada a objetos. El desarrollo de este juego se solicitó como trabajo final de asignatura a alumnos de primer año. El *Simon Says*[3] es un juego en el que la computadora emite una secuencia de luces de colores, y el jugador debe repetir esa secuencia exactamente en el mismo orden y está basado en un juego popular de niños muy difundido en todo el mundo. Su implementación como trabajo de práctica en los cursos de programación analizados permitió motivar a los estudiantes para investigar y aplicar elementos básicos de gestión de interfaces de usuario de alto nivel en Java, al mismo tiempo que se aplican técnicas de gestión elemental de gráficos por medio del marco de trabajo provisto por los profesores y ya citado en el *juego de cartas*. Elementos destacados: la lógica básica para programar el control general del juego es realmente sencilla y puede basarse por completo en el uso de un arreglo, convirtiéndose en una aplicación natural y concreta para esa estructura de datos. Además, visualmente el juego es compacto y estructurado, con lo cual resulta aceptablemente simple de diseñar aún si no se conoce demasiado sobre interfaces visuales.

e.) *El Juego de la Vida (de Conway)*: Con lenguaje Java y paradigma de programación orientada a objetos. El desarrollo de este juego se solicitó como trabajo final de asignatura a alumnos de primer año, en forma similar al *Simon Says*. El *Juego de la Vida*[4] Es lo que se conoce como un juego de *cero jugadores*: un juego que se auto-regula a partir de ciertas

[2] URL de consulta: http://en.wikipedia.org/wiki/Tic-tac-toe

[3] URL de consulta: http://www.mathsisfun.com/games/simon-says-game.html

[4] URL de consulta: http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

condiciones iniciales, sin intervención de jugadores externos. Fue creado por *John H. Conway* en 1970, y de allí su nombre. El juego simula entidades elementales que nacen, crecen, viven y mueren en función de ciertas reglas fijas. Causó furor entre los programadores por sus posibilidades de aplicación directa basadas en reglas simples y por la posibilidad de cambiar las reglas y estudiar otros patrones de comportamiento. Al igual que el *Simon Says*, fue muy útil para incentivar el estudio de interfaces de usuario basadas en ventanas y ampliar el dominio en gestión de gráficos. Elementos destacados: La lógica de control es sencilla. Visualmente es compacto y permite identificar patrones gráficos que incluso suelen recibir nombres propios. El tablero (o "hábitat") puede modelarse con un arreglo bidimensional y programarse como parte del framework provisto por los profesores, de forma que cada programador podría simplemente diseñar una clase que implemente una interfaz para indicar cómo quiere que se desarrollen las entidades en el hábitat.

f.) *El Tamagotchi*: Con lenguaje Java y paradigma de programación orientada a objetos. El trabajo se planteó para introducir a alumnos de primer año en técnicas de gestión elemental de gráficos y principios básicos de la POO. El *Tamagotchi*[5] es una mascota virtual creada en 1996 por Aki Maita y comercializada por la empresa Bandai. En su modelo básico, el *Tamagotchi* es un aparato con forma de huevo del tamaño de la palma de la mano que tiene una pantalla en blanco y negro pixelada, donde se puede ver una imagen de la mascota virtual. Debajo de la pantalla hay tres botones que permiten al usuario moverse por un menú para simular diversas actividades de interacción con la mascota. La experiencia de implementar una simulación del Tamagotchi resultó muy práctica para ayudar a fijar ideas y conceptos introductivos de la POO (encapsulamiento, principio de ocultamiento, creación y manipulación de objetos) y el dominio inicial en gestión de gráficos a través del framework provisto por los profesores. Elementos destacados: La lógica de implementación es directa, ya que sólo requiere una clase que represente a la mascota virtual y otra para modelar su ambiente. En el momento de plantearse la experiencia, no se requiere mayor tecnología de interfaz de usuario que la consola estándar para implementar un menú de opciones, más la consola de despliegue de gráficos provista por los profesores para desplegar imágenes de la mascota virtual que vayan cambiando a medida que se interactúa con ella.

g.) *La Generala*: Con lenguaje Java y paradigma de programación orientada a objetos. El trabajo se planteó para que los alumnos desarrollen aplicaciones con arreglos unidimensionales de tipos de datos simples. La *Generala*[6] es un juego en el que se utiliza un conjunto de dado y un cubilete. El número de jugadores es ilimitado, pero lo óptimo es de entre tres y cinco. El objetivo del juego es lograr el mayor puntaje, de acuerdo a una valorización establecida para cada "figura" o combinación posible de dados obtenida en el juego, llamada categoría. El trabajo consistió en una versión simplificada en la cual se generaba una tirada (con números aleatorios) y a partir de la misma el programa debía ofrecer las posibles categorías a las que aplicaba la misma además de graficar los dados usando el marco de trabajo para gestión de gráficos provisto por los profesores y ya mencionado en el ítem *juego de cartas*.

[5] URL de consulta: http://en.wikipedia.org/wiki/Tamagotchi

[6] URL de consulta: http://es.wikipedia.org/wiki/Generala

Los resultados superaron las expectativas, ya que los estudiantes introdujeron mejoras por sí mismos, tales como permitir la selección de algunos dados y volver a generar el lanzamiento, o incluso simular una partida con varios jugadores en simultáneo.

h.) *El Craps*: Con lenguaje Java y paradigma de programación orientada a objetos. Se planteó como un ejercicio de aula orientado a comprender e implementar el concepto de *relación de composición* entre clases y emplear la generación de números aleatorios dentro de un intervalo determinado. Las reglas del juego permiten además una interesante combinación de instrucciones condicionales y ciclos. El *Craps o Pase Inglés*[7] es un juego que consiste en realizar distintas apuestas al resultado que se obtendrá al lanzar dos dados. El ejercicio de aula consistió en implementar una versión similar a la variante conocida como "*línea de pase*" (aunque sin considerar un monto apostado): los alumnos debían determinar si el jugador ganó, perdió o debe volver a tirar en el primer lanzamiento, determinar el puntaje obtenido y gestionar los siguientes tiros hasta establecer el resultado final de la ronda. Desde el punto de vista del diseño orientado a objetos, se planteó una clase *Dado* para simular el lanzamiento aleatorio, y otra clase *Jugada* con dos dados como atributo (introduciendo así la relación de *composición*) con la correspondiente responsabilidad de evaluar el resultado final de la ronda. El ejemplo pudo resolverse dentro del tiempo de clase y los alumnos analizaron estadísticamente los resultados de la ejecución. Se dejó planteada la posibilidad de involucrar montos apostados y representación gráfica, así como reusar la lógica de dados en futuros trabajos prácticos (como en la implementación del juego de la *Generala*, anteriormente citado).

[7] URL de consulta: http://es.wikipedia.org/wiki/Craps

Discusión

La estrategia didáctica del diseño y desarrollo de juegos para motivar el aprendizaje de la programación es una idea muy difundida en todo el mundo. En este trabajo se han mostrado y analizado diversas experiencias locales, pero existen muchísimas otras iniciativas similares, con distintos niveles de desarrollo que van desde la experiencia llevada a cabo por docentes de la *Universidad de Kent* (Reino Unido) mediante la creación de modelos de cursos de enseñanza de la POO con Java, basados en los productos desarrollados por la propia Universidad de Kent como *BlueJ* (IDE para Java) y *GreenFoot* (IDE para desarrollo de juegos). El *proyecto BlueJ* (Kölling, 2013) dispone de una comunidad virtual abierta a profesores de programación de todo el mundo designada como *Blueroom*, y en el sitio de esa comunidad virtual está disponible el programa completo del curso citado más arriba[8] además de muchos otros recursos.

Por otra parte, el uso del *Tamagotchi* como elemento integrador para diversos fundamentos de POO es una idea generalizada y se puede citar al menos una experiencia adicional llevada a cabo en la *Universidad Veracruzana* (México), en la que un equipo de docentes de programación orientada a objetos y estructuras de datos planteó el desarrollo de materiales didácticos designados como *microunidades de competencia*, de tal forma que una de esas microunidades está basada en el desarrollo por parte de los alumnos de un programa que simule el funcionamiento del Tamagotchi (Ochoa Rivera & Contreras Vega, 2009).

El *Juego de la Vida (de Conway)* es un caso paradigmático en el que un juego se

[8] URL de consulta: http://www.bluej.org/. En este mismo sitio se incluyen enlaces para la descarga de los productos *BlueJ* y *Greenfoot*.

convierte en motivo de estudio permanente por parte de investigadores, docentes, estudiantes y/o simples aficionados a la programación. Los casos y experiencias alrededor de este juego son innumerables, al punto que existe al menos una comunidad virtual (Johnston, 2013) dedicada al estudio de este y otros juegos y autómatas celulares similares[9].

La experiencia de los *Juegos de Cartas* y sobre todo su marco de trabajo previo (aportado por los docentes a los alumnos) forma parte del planteo de un proyecto de investigación más amplio en el que participan los autores de este trabajo (Frittelli, et al., 2013). Esa y otras experiencias basadas en juegos para la enseñanza, promovieron la idea del planteo de un marco de trabajo o framework específico, para permitir que los estudiantes de diversas asignaturas de estructuras de datos y de inteligencia artificial puedan eventualmente participar en forma integrada en la programación de agentes virtuales para juegos soportados por ese framework. El trabajo con los juegos de cartas en particular, mostró la posibilidad de crear un framework prototípico, identificando clases abstractas e interfaces que los estudiantes pudiesen derivar o implementar para garantizar ciertos comportamientos y respuestas en el contexto de un juego preestablecido.

Conclusión

La selección y diseño de materiales de estudio para cursos de programación, así como el diseño de las estrategias didácticas a emplear en los propios cursos, es una tarea central en el trabajo docente y plantea desafíos profundos tanto desde el punto de vista pedagógico como desde la perspectiva técnica de la propia disciplina de la programación. El desafío pedagógico se basa en encontrar casos de análisis, conceptos, prácticas y situaciones que ya existan previamente en las redes conceptuales de los estudiantes (o que al menos, resulten cotidianos para ellos), de forma que la introducción de nuevos conocimientos pueda apoyarse en los que ya existen y se favorezca el aprendizaje significativo (y no memorístico). El desafío técnico consiste en diseñar aplicaciones prácticas eclécticas (ni demasiado sencillas ni demasiado complejas) que puedan ser presentadas a los alumnos para la introducción a la investigación y estudio de nuevos conocimientos, pero de forma que puedan ser desarrolladas por ellos mismos y que planteen desafíos lógicos y tecnológicos motivantes. Los resultados y posibilidades de las experiencias que se han mostrado en este trabajo, junto a muchas otras implementadas en distintos lugares del mundo, muestran que el desarrollo de juegos como estrategia didáctica para la enseñanza de la programación es una alternativa valiosa para enfrentar tanto el desafío pedagógico como el desafío técnico.

Referencias

[1]. Ahumada, P. (1990). Modelos de Evaluación y Evaluación de Programas. Valparaiso: Universidad Católica de Valparaiso.

[2]. Ausubel, D., Novak, J., & Hanesian, H. (2009). Psicología Educativa: Un punto de vista cognoscitivo (2da. ed.). México: Trillas.

[3]. Frittelli, V., Strub, A. M., Destéfanis, E., Steffolani, F., Teicher, R., Tartabini, M., et al. (2013). Motores de Juegos e Inteligencia Artificial para la Enseñanza. Workshop de Investigadores en Ciencias de la Computacion (WICC 2013). Paraná: Red UNCI.

[4]. Frittelli, V., Teicher, R., Tartabini, M., Fernández, J., & Bett, G. F. (2011). Gestión de Gráficos (en Java) para Asignaturas de Programación Introductiva. In Z. Cataldi, & F. Lage (Ed.), Libro de Artículos Presentados en I Jornada de Enseñanza de la Ingeniería - JEIN 2011. I, pp. 155-164. Buenos Aires: Universidad Tecnológica Nacional.

[5]. Johnston, N. (2013). ConwayLife.com - A community for Conway's Game of Life and related cellular automata. Retrieved Agosto 2013, from ConwayLife.com - [Official Website]: http://conwaylife.com/

[9] URL de consulta: http://conwaylife.com/

[6]. Kölling, M. (2013). BlueJ - The Interactive Java Environment. Retrieved Agosto 2013, from BlueJ - [Official Website]: http://www.bluej.org/

[7]. Langsam, Y., Augenstein, M., & Tenenbaum, A. (1997). Estructura de Datos con C y C++. México: Prentice Hall.

[8]. Malinowski, E. (1999). Enseñanza de Cursos de la Programación Orientada a Objetos para los Principiantes. (R. Herrera, Ed.) Ingeniería - Revista Semanal de la Universidad de Costa Rica, 9(1), 155-165.

[9]. Ochoa Rivera, C. A., & Contreras Vega, G. (2009). Proyecto AULA: Diseño y Aplicación de una Micro Unidad de Competencia para la Experiencia Educativa Algoritmos y Estructura de Datos II. Congreso Nacional y Congreso Internacional de Informática y Computación 2009 (CNCIIC 2009) (pp. 159-164). México: Asociación Nacional de Instituciones de Educación en Tecnologías de la Información (ANIEI).

[10]. Sedgewick, R. (1995). Algoritmos en C++. Reading: Addison Wesley - Díaz de Santos.

[11]. Weiss, M. A. (2000). Estructuras de Datos en Java - Compatible con Java 2. Madrid: Addison Wesley.

Datos de Contacto:

Ing. Valerio Frittelli.
UTN Córdoba - vfrittelli@gmail.com

Ing. Marcela Tartabini.
UTN Córdoba - mtartabini@gmail.com

Ing. Romina Teicher.
UTN Córdoba - rteicher@gmail.com

Ing. Felipe Steffolani.
UTN Córdoba - fsteffolani@gmail.com

Ing. Diego Serrano.
UTN Córdoba - diegojserrano@gmail.com

Ing. Julieta Fernández.
UTN Córdoba - jujulifer@gmail.com

Ing. Gustavo Federico Bett.
UTN Córdoba - gfbett@gmail.com

Ing. Ana María Strub.
UTN Córdoba - anastrub@gmail.com