Actas de las Jenui, vol. 3. 2018.
Páginas: 149-156

El código máquina mola

Francisco J. Gallego-Durán Rosana Satorre-Cuerda
Patricia Compañ-Rosique Carlos Villagrá-Arnedo
Cátedra Santander-UA de Transformación Digital
Universidad de Alicante
03690 San Vicente del Raspeig (Alicante)
{fjgallego,rosana.satorre,patricia.company,villagra}@ua.es

Resumen

Ciertas materias son especialmente difíciles o disgustan a los estudiantes. Incluso existen dichos populares donde las integrales, la física cuántica y otros conocimientos ejemplifican cuestiones muy complicadas. Estas materias suelen recibir críticas de estudiantes (en ocasiones, también de profesores), tener altas tasas de abandono o suspenso, tendencia a la copia o métodos alternativos para superarlas, etc. Lenguaje ensamblador y código máquina son dos buenos ejemplos. Es común escuchar críticas sobre ellos, su dificultad, la animadversión de profesores y estudiantes, o su escasa utilidad. ¿Es esta dificultad real e inherente a algunas materias? ¿Puede otro enfoque cambiar nuestra percepción?

Para responder a estas preguntas organizamos un curso de código máquina y ensamblador. Elaboramos todo el material priorizando la práctica sobre la teoría. Impartimos el curso a cien alumnos de nuevo ingreso una semana antes de comenzar las clases oficiales. Los resultados muestran que casi todos los estudiantes disfrutaron del curso. Las evidencias recogidas sugieren que no son las propiedades inherentes las que dificultan el aprendizaje, sino la forma en que el contenido es enfocado e impartido.

Abstract

Some subjects are especially difficult or displease students. There are even popular sayings where integrals, quantum physics and other knowledge exemplify very complicated questions. These subjects are often criticized by students (sometimes also by teachers), have high dropout or suspension rates, tendency to copy or alternative methods of overcoming them, etc. Assembler language and machine code are two good examples. It is common to hear criticism about them, their difficulty, the animosity of teachers and students, or their little usefulness. Is this difficulty real and inherent in some subjects? Can another approach change our perception?

To answer these questions we organize a machine code and assembler course. We elaborated all the material prioritising practice over theory. We give the course to one hundred new students a week before starting the official classes. The results show that almost all students enjoyed the course. The evidence suggests that it is not the inherent properties that hinder learning, but the way content is focused and imparted.

Palabras clave

Dificultad inherente, enfoque, metodología, código máquina, ensamblador

1. Introducción

Es común asumir que algunas materias son muy difíciles de enseñar debido a diferentes motivos. Matemáticas, Física o Química son conocidas por el temor que los estudiantes les tienen por su supuesta dificultad [2, 8, 13, 16]. En algunos casos llegan a producir fobias, como se refleja en [5] en relación a las Matemáticas. En estos casos, parece que el resultado no será bueno independientemente de la metodología empleada: elevadas tasas de abandono [3], tendencia a la copia [14], muchos suspensos, etc. Al ser cuestiones generalizadas, todo parece indicar que estas materias son arduas por su naturaleza, lo que dificulta que los estudiantes puedan valorarlas, entenderlas y dominarlas, no digamos ya disfrutarlas. Un ejemplo es la programación de ordenadores [1, 7, 10]. Enseñar programación a estudiantes de nuevo ingreso presenta todo tipo de dificultades, que llevan a múltiples enfoques [15, 18] y suelen producir frustración en los profesores [4].

La enseñanza de la programación suele seguir un enfoque algorítmico utilizando lenguajes de cada vez más alto nivel. Esta aproximación se justifica por la mayor cercanía de estos lenguajes al entendimiento humano y su mayor productividad. El objetivo es motivar a los estudiantes y allanar su camino hacia la comprensión,

bien sea utilizando gamificación o eliminando el contenido cuya dificultad parece inasumible y/o improductiva. En [6] se revisan algunos de los principales hitos históricos de esta evolución educativa hacia el alto nivel. A pesar de todos los esfuerzos, parece que programar sigue resultando muy difícil y/o aburrido a los estudiantes.

La programación es una materia con predominancia práctica: se interioriza y domina a través de la realización de la propia actividad. Resulta similar a habilidades como jugar al baloncesto: la técnica ortodoxa para lanzar a canasta se explica de forma teórica, pero sólo es posible adquirirla practicando. Así mismo, un mínimo de conocimiento es necesario para empezar: las reglas y objetivos son necesarios. Este factor resulta crucial en el aprendizaje: la motivación emerge más frecuentemente con objetivos de acción (querer encestar, jugar, ganar, etc.) que con objetivos de conocimiento [17].

Otro factor importante es la variabilidad individual del proceso de aprendizaje: no existe un único modo de aprender. Según describen distintas corrientes [9], podemos clasificar el aprendizaje en asociativo, aprendizaje por condicionamiento clásico, aprendizaje por observación e imitación, aprendizaje significativo, aprendizaje conceptual y aprendizaje acumulativo. Significativamente, muchas descripciones de estas clases de aprendizaje dotan de un papel protagonista al docente como encargado de transmitir conocimientos. Sin embargo, las metodologías más actuales conciben el proceso de aprendizaje como autónomo [19], con el docente ocupando un papel de asistente o facilitador.

Este trabajo busca respuestas a la siguiente pregunta de investigación: ¿Existe una complejidad inherente a determinadas materias que desmotiva y dificulta el aprendizaje? Al ser una pregunta muy amplia, este trabajo se limita a recabar evidencias en el ámbito de la programación en código máquina y lenguaje ensamblador. La pregunta concreta es: ¿Tienen el código máquina y el ensamblador una complejidad desmotivadora inherente? Dicho de otra manera, ¿Existe alguna aproximación motivadora para aprender código máquina y/o ensamblador?

Para responder a la pregunta, se ha diseñado un experimento de intervención: un curso basado en vídeos, con un enfoque innovador y una estructura similar a un MOOC [12]. La motivación de acción y la individualidad del aprendizaje han sido considerados para diseñar el enfoque innovador. El objetivo es comprobar si una materia considerada intrínsecamente desmotivadora puede resultar atractiva cambiando la forma de impartirla. Las evidencias recogidas permiten discutir si son las propiedades intrínsecas las que dificultan el aprendizaje.

El apartado 2 explica la estructura del curso, su predominancia práctica y su diseño para el aprendizaje autónomo, permitiendo a los estudiantes decidir los contenidos y su orden. La sección 3 detalla la metodología empleada para evaluar la validez del curso como experimento respecto a las preguntas de investigación. A continuación, la sección 4 presenta y analiza los resultados obtenidos, de los que se extraen las conclusiones que se resumen en el apartado 5.

2. Curso de ensamblador

Para obtener evidencias sobre la pregunta de investigación se diseñó un curso titulado “Programación y Videojuegos desde 0: Dominando el Ensamblador del Z80”. El curso introduce en la programación en código máquina y ensamblador sin asumir conocimientos previos. El contenido completo está disponible vía web[1].

2.1. Diseño

Dado que la intención era comprobar cómo un nuevo enfoque de enseñanza afecta al resultado, se establecieron unos objetivos fundamentales para el diseño del curso:

1. Inicio desde cero, sin conocimientos previos.
2. Comenzar en código máquina: el nivel más bajo, considerado tedioso, críptico y nada interesante.
3. Enfocar a los estudiantes a la acción: práctica primero, teoría después.
4. Objetivo de acción principal: crear un videojuego.
5. Dividir el objetivo en subobjetivos de acción.
6. Permitir elecciones y ritmos individualizados.

Este planteamiento perseguía motivar a los estudiantes a través de tareas creativas de programación, considerando las premisas de acción y aprendizaje autónomo individual.

Los objetivos determinaron que el curso debía ser autocontenido, para permitir ritmos y elecciones personalizados. Por tanto, se elaboró el contenido en vídeo. Eso garantizaría acceso a los contenidos en el tiempo y orden que prefirieran.

Como crear un videojuego es un objetivo complejo, resultó muy importante el diseño de subobjetivos. Los subobjetivos deben tener entidad individual para ser motivadores, y deben aportar conocimiento y capacidades progresivamente al objetivo principal. Esto dio lugar a una estructuración en niveles, con un desafío como guía de contenidos de cada nivel, y con retos como actividades individuales que preparan para el desafío.

Se diseñaron vídeos para cada nivel divididos en 2 categorías: teoría y práctica. Los vídeos prácticos pre-

[1] `https://dez80.byterealms.com` Usuario: `curso` Contraseña: `Ensamblador4TheWin`. Accedido 9/2/2018.

sentan los retos y desafíos a superar, y muestran algunos ejemplos como guía para empezar a realizarlos. Se indica a los estudiantes que el objetivo es superar los desafíos y pasar de nivel. Esto implica que no es necesario que vean todos los vídeos: sólo los que les interesen. Los vídeos de teoría quedan en un plano secundario. Pasan a ser una ayuda para realizar los retos y desafíos, que los estudiantes deben consultar cuando perciban que les es útil. Este es un enfoque parecido a la clase invertida que los autores consideramos como aportación innovadora en los cursos MOOC, pues no tenemos conocimiento de otros con este mismo enfoque.

Por último, se consideró adecuado para un contenido que comienza desde cero utilizar un ordenador más sencillo que los actuales. Se optó por utilizar un Amstrad CPC 464, con 64K de RAM y procesador Z80, del que existen emuladores muy exactos, como WinAPE[2]. Este emulador incluye un depurador que facilita el rápido entendimiento de la máquina, la ejecución paso a paso e incluso la edición manual de la memoria, permitiendo introducir programas en código máquina. Por otra parte, los conocimientos básicos de arquitectura de procesadores que se pueden adquirir con el Z80 son exportables a cualquier procesador moderno, constituyendo un buen punto de partida.

2.2. Estructura y contenidos

El curso está estructurado en niveles. Cada nivel contiene un desafío final para superar. Además, cada nivel contiene una serie de retos que preparan al estudiante para enfrentarse al desafío. Los vídeos prácticos contienen también ejemplos para entender cómo realizar los retos y desafíos.

Todos los retos y desafíos son autoevaluables. El estudiante sabe cuando ha superado un reto o desafío porque su programa realiza el objetivo marcado. El formato es de solución abierta: hay incontables posibles soluciones para cada reto. Lo importante no es encontrar una solución concreta, sino desarrollar una solución propia. Esta filosofía es indicada al estudiante a través de los vídeos.

Una consecuencia relevante del diseño es el orden resultante para los contenidos. Los enfoques clásicos tienden a priorizar bloques teóricos en orden constructivo, bajo la premisa de conocer antes hacer. Al priorizar la práctica, aparecen curiosidades como conocer antes punteros que variables, codificaciones y datos sin el concepto de tipo, conversiones numéricas con utilidad inmediata, ejecución paso a paso y depuración desde el principio, gestión básica de memoria antes que rutinas y funciones, etc. Creemos relevante experimentar reorganizando contenidos guiados por la acción práctica en lugar del requerimiento teórico, ya que puede tener influencia en la motivación.

Se han elaborado más de 50 vídeos para 4 niveles con los siguientes contenidos:

Nivel 0. Introducción, filosofía, estructura, evaluación y contenidos.

Nivel 1. Dibujar un *sprite*[3] en código máquina.

- *Retos*: Aprender a colorear píxeles individuales, filas y columnas de *píxeles*, pintar en distintas zonas de la memoria de vídeo/pantalla, codificar un *sprite*.
- *Contenidos*: Uso e instalación de WinAPE, introducción al Amstrad CPC 464, memoria y procesador, ejecución, binario, hexadecimal, conversiones numéricas, memoria de vídeo, formato de codificación de píxeles, registros del procesador, ejecución paso a paso, instrucciones en código máquina para cargar y mover datos.

Nivel 2. Animar un *sprite* en código máquina.

- *Retos*: Animar píxeles individuales cambiando de color y, después, cambiando de posición, usar múltiples fotogramas, realizar grandes dibujos patronizados con estructuras de repetición.
- *Contenidos*: Interrupciones y control básico del tiempo, funcionamiento del monitor (*raster scan*[4]), números positivos y negativos en binario y hexadecimal, repeticiones, contadores y bucles, punteros, gestión básica de memoria y almacenamiento, variables en memoria, dibujado con *tiles*[5](Baldosas).

Nivel 3. Crear un videojuego en ensamblador

- *Retos*: Ensamblar código por primera vez, animaciones con movimiento, controlar la acción con teclado, primer minijuego simple, mejorar el control desde teclado, crear un videojuego propio en ensamblador.
- *Contenidos*: Lenguaje ensamblador, como funciona el programa ensamblador, rutinas y llamadas, la pila de programa, funciones, saltos condicionales, introducción al *firmware*, lectura de teclado, manejo de variables y memoria, depuración con puntos de ruptura (*breakpoints*).

2.3. Desarrollo del curso

Se invitó a participar a los 250 estudiantes de nuevo ingreso en las titulaciones de Grado en Ingeniería Informática y Grado en Ingeniería Multimedia. Entre los invitados, se seleccionaron al azar 100 estudiantes, con una proporción definida entre ambos grados ajustada a

[2] `http://www.winape.net`. Accedido 9/2/2018.

[3] En videojuegos, dibujo cuadrado asociado a personaje u objeto.

[4] Paso del haz de electrones por la pantalla de tubo de rayos catódicos, refrescando la imagen.

[5] *Sprites* usados como patrón para formar dibujos por repetición.

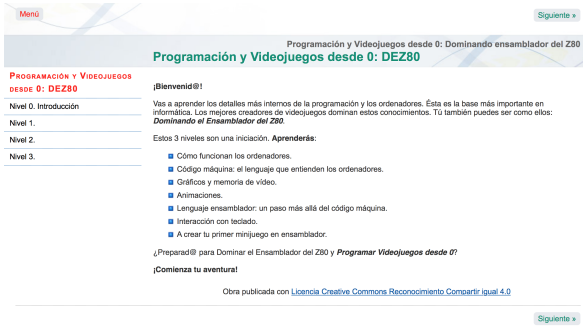

Figura 1: Página web del curso

Figura 2: Video que presenta el primer desafío

los *numerus clausus* de las titulaciones (65 informática y 35 multimedia).

El curso se realizó de forma presencial, la semana previa al inicio oficial del curso académico, del 4 al 8 de septiembre de 2017. Las 20 horas se distribuyeron en 4 horas diarias de lunes a viernes, de 16:00h a 20:00h. Las 4 horas incluían un descanso de 20 minutos, haciendo un total de 1 hora y 40 minutos en los 5 días. Es importante considerar que los estudiantes tenían también hasta 4 horas de clases de cursos cero por la mañana.

El curso empezó con una sesión plenaria de Introducción-Motivación[6] de 30 minutos de duración. Se explicó a los estudiantes lo importantes que eran los conocimientos del curso para su futuro profesional, y se les explicó cómo adquirirlos mediante la práctica, evitando las típicas frustraciones del proceso de aprendizaje. Sobre todo, se hizo mucho énfasis en que equivocarse es un paso necesario en el aprendizaje y que no debían pensar en hacer las cosas bien a la primera, sino en insistir hasta conseguir hacerlas.

Los estudiantes fueron distribuidos en 4 aulas contiguas de prácticas con ordenadores. Se les proporcionó acceso a la web del curso (figura 1) y se pusieron directamente a trabajar. La mayor parte del tiempo, los estudiantes veían los vídeos (figura 2) y realizaban las tareas de forma autónoma. Cada aula contaba con un profesor que se dedicaba a ayudar individualmente a los estudiantes con las dudas y problemas que les surgían.

Para dinamizar el trabajo del curso se incluyeron 2 actividades principales:

- El tercer día se les motivó a publicar en *twitter* imágenes sobre los logros que fueran consiguiendo, bajo el *hashtag* #dez80[7]. Se aprovechó la ocasión para indicarles que algunos profesionales no creían que estuvieran aprendiendo código máquina sin recibir algo a cambio (como créditos, por ejemplo). Los estudiantes comenzaron a publicar en *twitter* sus logros y consiguieron que varios profesionales del mundo del videojuego los comentaran e interactuaran.
- La segunda mitad del cuarto día (2 horas) se dedicó a un evento programado. En sesión plenaria en el salón de actos se realizó una imitación del concurso televisivo "Quien quiere ser millonario"[8]. Se prepararon dos baterías de preguntas y varios estudiantes salieron al escenario cómo si fueran concursantes. Se utilizaron las mismas reglas que en el concurso. Además, las preguntas eran respondidas siempre por todos los estudiantes a través de la aplicación "Socrative". Estas respuestas eran utilizadas también para simular el comodín del público.

Tras descansos y actividades, el tiempo efectivo de trabajo fue de aproximadamente 15 a 16 horas.

3. Metodología

Puesto que el curso era autoevaluable, la obtención de evidencias del experimento se ha basado en dos encuestas realizadas a los estudiantes [9].

La primera encuesta se realizó durante la inscripción. Esta encuesta se diseñó para conocer el perfil de los estudiantes. Los datos fundamentales que se obtuvieron de ella son relativos a los conocimientos previos de programación y la forma en que los habían adquirido (mediante educación formal/reglada o de forma autodidacta). También se obtuvieron datos relativos a los lenguajes de programación que conocían, el tiempo que llevan programando y su afinidad con la programación en general (si les gusta programar y qué tipos de programas les gustaría crear). Lógicamente, esta in-

[6] https://vertice.cpd.ua.es/182519. Accedido 9/2/2018.

[7] https://twitter.com/hashtag/dez80?f=tweets&vertical=default. Accedido 9/2/2018.

[8] https://es.wikipedia.org/wiki/%C2%BFQui%C3%A9n_quiere_ser_millonario%3F. Accedido 9/2/2018.

[9] http://hdl.handle.net/10045/73430. Accedido 9/2/2018

formación sólo fue obtenida del grupo de estudiantes que indicaron tener conocimientos previos de programación.

Por su parte, la segunda encuesta tuvo lugar el último día del curso. Ésta se focalizó en tres puntos principales: la asistencia, el aprendizaje y la satisfacción. Al tratarse de una encuesta, los datos de estos tres puntos son siempre en valor percibido por los estudiantes. Puesto que la pregunta de investigación de partida tiene mucho que ver con la percepción de los estudiantes, resulta apropiado analizar los valores percibidos.

Se les preguntaba los días asistidos y horas dedicadas, el nivel alcanzando y los conceptos que consideraban aprendidos, de forma pormenorizada. De igual forma, se les preguntó su percepción de dificultad de todos los retos, también de forma pormenorizada. Para la dificultad de los retos se utilizó una escala de Likert [11] (muy fácil, fácil, normal, difícil, muy difícil), con el añadido de que pudieran responder también "No lo he hecho" para los retos que no habían alcanzado. También se les preguntó los conocimientos en los que les gustaría profundizar y, finalmente, se les reservó un espacio para respuestas abiertas en las que expresar su valoración global de los distintos niveles y el curso.

4. Resultados

Los primeros resultados interesantes aparecen en una de las preguntas más simples. Como muestra la figura 3, casi la mitad de los estudiantes afirma tener conocimientos de programación antes de entrar a la universidad. Estos conocimientos previos provenían en un 53 % de la educación reglada (primaria y secundaria), 24 % de forma autodidacta pura, 17 % mezcla de educación reglada y autodidacta y tan sólo un 6 % de cursos de formación.

En cuanto a lenguajes conocidos, el 67 % mencionó conocer *C/C++*, el 31 % *Scratch*, el 20 % *Java*, el 15 % *Javascript*, un 9 % conocía *Python* y *Arduino*, y finalmente un 2 % mencionó *C#*. Los porcentajes son solapados porque algunos estudiantes afirmaron conocer más de un lenguaje. A los profesores nos sorprendió bastante esta distribución tan favorable a *C/C++*. Pese a ello, es destacable que el 53 % indicaban 1 o menos años de experiencia programando, frente al 40 % que indicaba 2 años, y tan sólo un 7 % que decía tener 3 o más años. En otras palabras, la mitad tan sólo han recibido 1 año de formación en programación, y casi la otra mitad han recibido 2, con algunas excepciones que llevan más tiempo. Curiosamente, sólo uno de los estudiantes mencionó ensamblador entre diversos lenguajes conocidos. Estos resultados son una muestra de como el ensamblador no es considerado un lenguaje adecuado para la enseñanza, ni popular, tal como habíamos supuesto inicialmente.

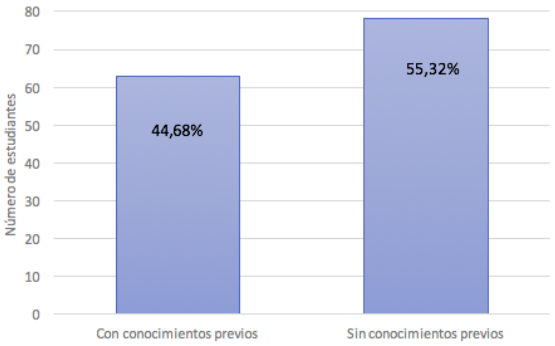

Figura 3: Conocimientos previos

La figura 4 muestra la lista con todos los conceptos considerados en el curso y el número de estudiantes que indican haber aprendido cada uno de ellos. Los conceptos iniciales del curso, correspondientes al nivel 1, han sido adquiridos por la gran mayoría de los estudiantes[10]. Conforme los conceptos se acercan más al final del curso, son más los estudiantes que no indican haberlos adquirido. El resultado es esperable en cuando a su distribución, y muy relevante respecto a la pregunta inicial. Aproximadamente la mitad de los estudiantes indica haber aprendido a "Realizar animaciones" o "Crear bucles con saltos condicionales", que son parte de los conceptos principales del nivel 2. Sobre un tercio de los estudiantes indican entender cómo funcionan los números positivos en binario y hexadecimal, saber usar variables (guardando datos en memoria), saber cómo funciona el *raster* de pantalla e incluso entender cómo se traduce el ensamblador a código máquina. Nuestra opinión como profesores es que son resultados muy valiosos, teniendo en cuenta que se producen en sólo 15 a 16 horas partiendo de cero respecto a estos conocimientos.

La figura 5 nos da una idea de la influencia de los conocimientos previos respecto a la cantidad total de conceptos que los estudiantes perciben haber aprendido. Sería esperable que las barras naranja fueran mayores en la mitad izquierda de la gráfica. Esto indicaría que hay más estudiantes sin conocimientos previos que reconocen haber aprendido pocos conceptos. Sin embargo, esto sucede sólo en el tramo inferior (de 4 a 8 conceptos). Las diferencias entre ambos grupos no parecen ser significativas, con un muy ligero favoritismo hacia los estudiantes con conocimientos previos. Sería interesante un estudio más detallado por lenguajes, experiencia u otros factores para interpretar mejor los resultados, aunque no es la cuestión principal de este trabajo.

Los conceptos aprendidos parecen indicar que casi

[10]La encuesta final sólo fue respondida por 64 de los 100 asistentes.

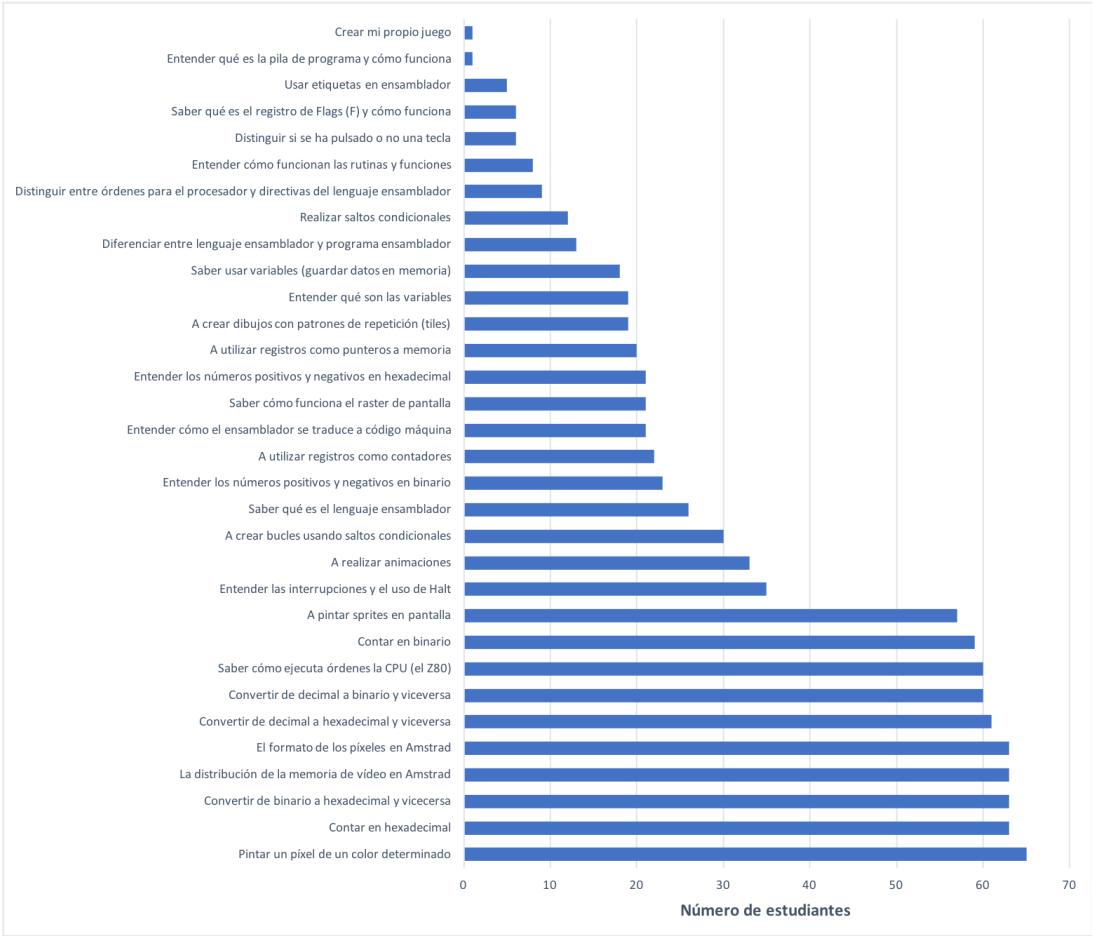

Figura 4: Conceptos adquiridos por los estudiantes

todos los estudiantes han superado el 30 % de los objetivos (nivel 1), un 30-35 % han superado el 65 % de los objetivos (nivel 2) y aproximadamente un 5 % han superado todo. Estos resultados serían equiparables a los de cualquier asignatura estándar, no especialmente difícil. Por tanto, parecen sugerir que los estudiantes no han tenido dificultades especiales al enfrentarse al código máquina y el ensamblador, más allá del poco tiempo de duración.

La figura 6 apoya esta línea argumental. En ella podemos ver la dificultad media percibida por los estudiantes al realizar cada reto. Los 6 primeros retos corresponden al nivel 1, los 5 siguientes al nivel 2. No se incluyen los siguientes retos del nivel 3 porque la muestra se reduce mucho y deja de ser representativa.

De forma esperable, cada reto es percibido como ligeramente más difícil que los anteriores, excluyendo el efecto de la varianza. Muy relevantemente, este crecimiento está entre los valores 2 (fácil) y 3 (normal). Esto evidencia que la dificultad percibida por los estudiantes es normal en general e incluso fácil al principio y según los retos. Estos datos chocan frontalmente con

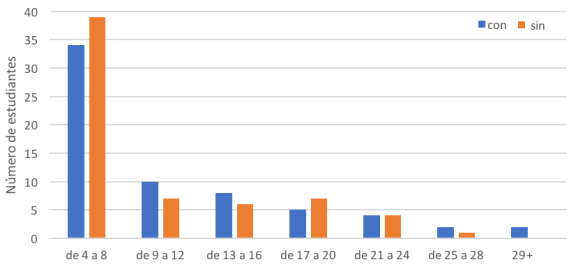

Figura 5: Cantidad de conceptos adquiridos

la idea de que el código máquina (contenido de los 2 primeros niveles) es intrínsecamente difícil o desmotivador. Los estudiantes parecen haberlo realizado con normalidad e incluso con comodidad, lo que favorece la hipótesis de que motivación y dificultad estén asociadas al método de enseñanza y no al contenido.

Considerando la motivación de los estudiantes, una apreciación relevante: la mayoría prefirieron continuar con sus programas durante los descansos, terminando

cuando cerrábamos el aula. Esta actitud nos sorprendió muy gratamente a los profesores, pues no esperábamos tanto interés. Por otra parte, más del 80 % indicó que le gustaría seguir aprendiendo ensamblador, funcionamiento interno de los ordenadores y cómo los lenguajes son traducidos en ensamblador y código máquina. Además, más del 90 % indicó querer aprender más sobre otros ordenadores y consolas, programación de videojuegos y *sprites*, píxeles y gráficos.

Además, la práctica totalidad de los comentarios libres sobre el curso fueron muy positivos. Muchos estudiantes incluso nos dieron las gracias de forma personal y nos dijeron que les había gustado mucho el curso. Incluso sus *tuits* publicados fueron muy positivos. La figura 7 muestra dos *tuits* de ejemplo. A continuación citamos algunos fragmentos:

- "Un curso genial para empezar a programar videojuegos, tal vez un poco de guía al comienzo no habría estado mal pero en resumen me ha encantado"
- "Gracias a este curso he aprendido a programar en código máquina y a moverme fluidamente por el sistema binario y hexadecimal. Es curioso comprender como funciona un ordenador de hace más de 30 años"
- "Sí, me ha resultado muy gratificante de realizar. Que los retos estén estructurados en forma de misiones como las de los videojuegos, con bonificaciones y eso, hace el curso más ameno. Aunque tienes que prestar dedicación y muchas horas, he pasado la semana entera delante de mi ordenador, yo lo recomendaría a cualquier persona que quisiera aprender como se programa, como funciona de verdad, incluso aunque no tenga conocimientos previos de programación."
- "Me parece muy interesante el enfoque del curso a partir de la base de los videojuegos en pc. Es complejo el principio aunque poco a poco se coge el ritmo. La parte negativa es que sean 4h seguidas y todo en una misma semana, llega a hacerse tedioso, cambiaría a 2/2,30h y en 2 semanas para ser más productivo. En general muy contento y lo recomiendo para comenzar la programación[...]"
- "El curso me ha parecido muy interesante ya que el ámbito de la programación, dentro de la variedad del mundillo de la informática, me encanta. Pero por otro lado una semana me ha parecido un período demasiado corto como para poder completar el curso. Gracias por la oportunidad, me parece que ha sido la mejor forma de empezar en la Universidad."

En general, tanto las opiniones cualitativas como cuantitativas sugieren que han disfrutado del curso, e incluso que les gustaría continuar estudiando sobre código

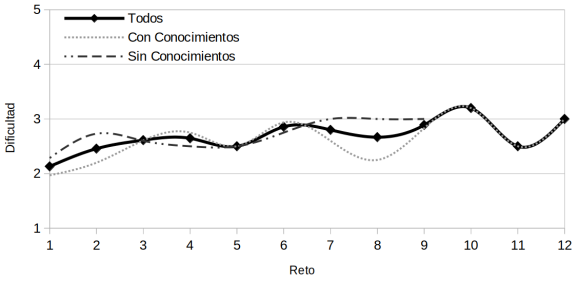

Figura 6: Dificultad media percibida

máquina y ensamblador. Estas evidencias también apoyan la hipótesis de que dificultad y desmotivación no son propiedades inherentes de estas materias.

5. Conclusiones

Este trabajo presenta una experiencia docente de enseñanza de código máquina y ensamblador a estudiantes de nuevo ingreso en titulaciones de Grado en Ingeniería en Informática y Grado en Ingeniería Multimedia. El objetivo es comprobar si estas materias son inherentemente difíciles de aprendizaje o desmotivadoras.

Como experiencia de intervención se ha diseñado un curso presencial de 20 horas, basado en vídeos y con apoyo de profesores. El curso presenta como aspecto innovador su focalización en la práctica, dejando la teoría como subsidiaria de las necesidades de acción. Cien estudiantes han participando y se han obtenido evidencias mediante encuestas previa y posterior.

Las evidencias obtenidas indican que los estudiantes han aprovechado la experiencia con unos resultados comparables a los de cualquier asignatura normal, sin características especiales de dificultad. Así mismo, los datos sugieren que no sólo han aprendido código máquina y ensamblador, sino que lo han hecho disfrutando de la experiencia, y quedando con ganas de aprender más. Todas las evidencias obtenidas contradicen la hipótesis que el código máquina y/o el ensamblador son inherentemente difíciles o desmotivadores. Por tanto, creemos probable que la percepción genérica de dificultad y desmotivación se deba a los métodos de enseñanza utilizados hasta ahora.

Se requiere más evidencia para corroborar estas conclusiones y ver si son aplicables a otras materias consideradas inherentemente difíciles o desmotivadoras.

Referencias

[1] Jens Bennedsen y Michael E Caspersen (2007). Failure rates in introductory programming. *AcM SIGcSE Bulletin*, *39*(2), 32–36.

Figura 7: *Tuits* de dos participantes del curso

[2] L. Cardellini (2012). Chemistry: Why the Subject is Difficult? *Educación Química*, *23*, 305 – 310.

[3] Agustín Cernuda del Río (2013). Un estudio sobre el absentismo y el abandono en asignaturas de programación. *ReVisión*, *6*(1).

[4] A. Cernuda-del Río (2017). Todavía no sabemos enseñar programación. *ReVisión*, *10*(1), 13 – 15.

[5] Silenne Fernández Rodríguez (2016). Evidencias de fobia, miedo o rechazo hacia la matemática en estudiantes de décimo año del colegio el carmen de alajuela. Master's thesis, Universidad estatal a Distancia.

[6] Francisco J Gallego-Durán, Rosana Satorre Cuerda, Patricia Compañ-Rosique, y Carlos J Villagrá-Arnedo (2018). Explicando el bajo nivel de programación de los estudiantes. *ReVisión*.

[7] Anabela Gomes y António José Mendes (2007). Learning to program-difficulties and solutions. In *International Conference on Engineering Education–ICEE*, volume 2007.

[8] H Kim y B Lee (2006). Why do secondary students perceive physics is uninteresting and difficult. *New Physics: Sae Mulli*, *52*(6), 521–529.

[9] Klausmeier, Herbert J. y Goodwin William (1997). *Enciclopedia de Psicología Educativa : Aprendizaje, Habilidades Humanas y Conducta*. Oxford University Press.

[10] Essi Lahtinen, Kirsti Ala-Mutka, y Hannu-Matti Järvinen (2005). A study of the difficulties of novice programmers. *SIGCSE Bull.*, *37*(3), 14–18.

[11] R. Likert (1932). A technique for the measurement of attitudes. *Archives of Psychology*, *22*(140), 1–55.

[12] Andrés Marzal Varó (2014). Próxima estación: Mooc. *ReVisión*, *7*(1).

[13] John Roche (1995). Why is physics so difficult? *Physics World*, *8*(6), 17.

[14] Juan Carlos Rodríguez-del Pino, Enrique Rubio Royo, y Zenón Hernández Figueroa (2012). El plagio de prácticas de programación: análisis de diez años de experiencia.

[15] Miguel Ángel Rubio, Carolina Mañoso, Rocío Romero Zaliz, y P Ángel (2014). Uso de las plataformas lego y arduino en la enseñanza de la programación. *Jornadas de Enseñanza Universitaria de la Informática (20es: 2014: Oviedo)*.

[16] Quintana Silva y Maria de Lourdes. La influencia de la fobia hacia la matemática en el desarrollo del pensameinto logico.

[17] Stella Maris Vázquez (2009). Motivación y voluntad. *Revista de Psicología*, *27*(2).

[18] Jorge A Villalobos y Nadya A Calderón (2009). Proyecto cupi2: un enfoque multidimensional frente al problema de enseñar y aprender a programar. *Revista de Investigaciones UNAD*, *8*(2), 45–64.

[19] Gloria María Sierra Villamil, Luz Marina Ramírez Hernández, William Fernando Rodríguez Torres, y Nohora Elsa Rodríguez Peña (2016). Las competencias pedagógicas del tutor virtual en un modelo de aprendizaje autónomo y de aprendizaje colaborativo. *Virtu@ lmente*, *3*(2), 55–83.