UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja
Máster Universitario en e-Learning y Redes Sociales

Desarrollo de una Herramienta para la Enseñanza de la Programación a Niños a Través de la Música

Trabajo Fin de Máster

Línea 7. Diseño de juegos y simulaciones educativos virtuales

Presentado por: Martínez Contador, Daniel

Director/a: Ruíz Rube, Iván

Ciudad: Murcia

Fecha: 25 de julio de 2014

Resumen

Motivados por la potenciación en la educación infantil de las materias STEM (sobre ciencia, tecnología, ingeniería y matemáticas) en EEUU ha habido una proliferación de herramientas software y hardware para facilitar el aprendizaje de la programación por niños. En este trabajo se desarrolló un prototipo de herramienta software innovadora, *APaM: Aprende Programación Haciendo Música*, para facilitar el aprendizaje de la programación por niños con la música como elemento conductor. Para ello, se estudiaron las motivaciones y beneficios para los niños de aprender programación, se analizaron críticamente las herramientas ya existentes y sus características, y se investigaron las tendencias en el mundo de la programación de usuario final, buscando puntos de mejora y nuevos enfoques para hacer esta herramienta innovadora. Finalmente, se discutieron las decisiones de diseño adoptadas, se detalló la implementación y se validó la herramienta con expertos y usuarios finales, mostrando las posibilidades de APaM.

Palabras Clave: software, aprendizaje, programación, música, niños

Abstract

Motivated by the strong support of STEM fields in USA education, there has been a proliferation of software and hardware tools that aim to ease the learning of programming by children. In this work we develop a prototype of an innovative software tool, *APaM: Aprende Programación Haciendo Música* (Learn Programming Doing Music), easing the learning of programming by children with the music as a motivational element. To achieve this, we studied the motivations and benefits for children of learning programming, we analyzed already existing tools and their characteristics and we researched current trends in end-user programming, looking for improvement points and new scopes to make this tool really innovative. Finally, we discuss the adopted design decisions, we detail the implementations process and we validate the tool with experts and final users, showing the possibilities of APaM.

Keywords: software, learning, programming, music, children

Índice de contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

En este trabajo fin de máster (TFM) se desarrollado una nueva herramienta para facilitar el aprendizaje de la programación, integrándose en el ecosistema existente de este tipo de herramientas.

En este capítulo introductorio se ha analizado el contexto y las motivaciones existentes, la necesidad de otras herramientas, y se ha desglosado la estructura de este documento.

1.1. Contextualización

En la actualidad se pueden encontrar un gran número de herramientas destinadas a facilitar el aprendizaje de la programación, siendo muy destacables las orientadas a niños. La mayoría de estas vienen de los Estados Unidos de América, donde desde hace más de una década se está potenciando toda la enseñanza en las materias *STEM*, ciencia, tecnología, ingeniería y matemáticas (Gonzalez & Kuenzi, 2012), desde educación primaria por la falta de profesionales especializados en esos campos. En un discurso reciente el Presidente Barak Obama hacía un alegato dirigido a los más jóvenes para que estos aprendieran a programar.[1] En la iniciativa *Una hora de código*[2], algunas de las más importantes personalidades del mundo de las tecnologías de la información, como Bill Gates (fundador de Microsoft), Steve Jobs (de Apple, a título póstumo, de grabaciones de archivo), Jack Dorsey (de Twitter), etc., se han unido para promover el aprendizaje de programación en las escuelas de EEUU. Las profesiones relacionadas con la computación (ciencias de la computación, ingeniería y desarrollo software, programación de aplicaciones, etc.) están entre las profesiones con mejor sueldo de entrada, mayor demanda en los EEUU y de mayor crecimiento en los próximos años (U.S. Department of Labor, 2014).

Aunque se critica algunas de estas iniciativas por oportunistas y orientadas a la "educación" de nuevos consumidores y creación de demanda de productos tecnológicos por parte de particulares y escuelas, o por un planteamiento político de fomentar el estudio de tecnología y ciencias de la información (Dvorak, 2013), es innegable la relevancia que tiene en nuestra sociedad el *alfabetismo tecnológico* como capacitación transversal a muchas capacidades en la vida, desde usar programas de mensajería en el móvil hasta hacer los

1 Blog de la Casa Blanca de los EEUU. http://www.whitehouse.gov/blog/2013/12/09/don-t-just-play-your-phone-program-it

2 One Hour of Code. http://code.org

impuestos en el ordenador. Pero esto solo no justifica la necesidad de *saber programar*, y de aprender *desde niños*.

Así pues, aparte de los intereses políticos para desarrollar las profesiones relacionadas con este campo y del valor puramente curricular de incorporar programación en el temario educativo, ¿qué motivos hay para enseñar programar a niños?

Aprender programación es posible para niños de edades tan tempranas como 10 años e incluso más jóvenes, disfrutando al mismo tiempo y resultándoles motivador. Los padres también perciben aspectos positivos como la mejora en el pensamiento lógico o el acercamiento a las ciencias de la computación. Así que, *«¿por qué no?»* (JMC Lin, Yen, Yang, & Chen, 2005).

La práctica de la programación permite desarrollar mejor las técnicas necesarias para enfrentarse a problemas de la vida real que el aprendizaje tradicional de las matemáticas no contempla. Los problemas de matemáticas están enfocados a un tipo de problema determinista, como por ejemplo «cuál es el área de una habitación cuadrada de doce baldosas de un metro de lado». Las situaciones de la vida real presentan problemas con un espacio de soluciones más amplio: «cómo debería moverse una mariposa para recorrer un número de habitaciones sin pasar varias veces por el mismo lugar». Experimentar con herramientas como Scratch[3] y otras de aprendizaje de computación, permite desarrollar mejor las técnicas necesarias para enfrentarse a este tipo de problemas más frecuentes en la vida real, y estas herramientas pueden ser apropiadas para integrarse en el aula con alumnos de primaria o secundaria (Olabe, Basogain, Olabe, Maíz, & Castaño, 2014).

Además de esto, los lenguajes de programación pueden ser la base para comprender mejor el conocimiento basado en conceptos abstractos y objetos lógicos, como lo es el latín para comprender las lenguas romances (Soloway, 1993). Así, a través de la programación es posible capacitar para la comprensión de conceptos abstractos.

También se ha demostrado que el aprendizaje y práctica guiados de la programación en el entorno educativo puede tener un impacto positivo en el desarrollo de la creatividad. El uso con este fin de la programación con Logo[4] ha sido probado con éxito en alumnos de primaria y los mejores resultados se obtienen con la incorporación temprana en el aula (Pardamean, 2014).

3 Scratch, http://scratch.mit.edu
4 The Logo Foundation, http://el.media.mit.edu/logo-foundation/index.html

Por último, y no menos importante, programar es divertido. Unos de los aspectos más importantes de la diversión en el juego es el reconocimiento de patrones (Koster, 2004) somos seres inteligentes y el reconocer patrones nos ejercita, estimula y finalmente divierte. En la programación hay un proceso cíclico de reconocimiento de patrones y su replicación en el dominio del lenguaje de programación que se está usando. Y más aún: uno de los principios de la programación es la reutilización, esto es, no programar dos veces lo mismo reusando lo programado la primera vez (y surge la búsqueda y aplicación de patrones de reutilización: *¡metaprogramación!*). La minimización de la repetición de patrones está dentro de la misma naturaleza de la programación, eliminando las tareas repetitivas y facilitando aún más entrar en el *flujo* descrito por Csíkszentmihályi (Csikszentmihalyi, 1990; Pilke, 2004).

En este trabajo se describe la construcción de un prototipo de herramienta para el aprendizaje de programación mediante la música. Tener este componente musical como centro del dominio de aplicación de la herramienta y elemento de ambientación general, una decisión innovadora, sirve para simplificar la herramienta sin perder profundidad y para involucrar al estudiante como usuario.

1.2. Metodología

Este TFM se ha realizado en varias etapas. En la primera etapa se ha realizado una investigación para concretar qué tipo de herramienta hacer. En primer lugar se han estudiado algunas de las herramientas e iniciativas más notables por su impacto, historia o enfoque usadas para el aprendizaje de la programación. Se han clasificado e identificado las características más deseables o menos explotadas hasta el momento en este tipo de herramienta. Se han analizado también las tendencias actuales en programación y su posible evolución. Con esta información se han definido la necesidad y características de una nueva herramienta.

En la segunda etapa se ha desarrollado un prototipo básico con una funcionalidad limitada y con un juego reducido de contenidos educativos de soporte para la enseñanza. Sería muy ambicioso esperar que de un TFM pueda surgir una herramienta completa capaz de competir en funciones con algunas de las más famosas, como Alice[5] o Scratch, desarrolladas a lo largo de años por equipos con amplia experiencia de grandes universidades. Sin embargo, se espera conseguir una herramienta que facilite el aprendizaje

5 Alice, http://www.alice.org

de la programación presentando un enfoque alternativo a las herramientas ya existentes, a ser posible tanto en el método educativo como en el tipo de cosas que se pueden hacer con la herramienta; con una baja barrera de entrada a educadores y niños, facilitando su inserción en un currículum; que resulte motivador para su uso en el aula o incluso de forma individual; flexible, adaptable y fácil de extender; y con un nivel de usabilidad adecuado, de tal forma que pueda ser desplegado y usado con una infraestructura tecnológica reducida.

En el desarrollo se ha seguido una metodología de inspiración ágil[6], con las adaptaciones que derivan de que APaM ha sido desarrollado por una sola persona. La aplicabilidad de las metodologías ágiles está probada en entornos caracterizados por la variabilidad, alta incertidumbre, especificaciones poco definidas, riesgo de fracaso, etc., riesgos que se han descrito con detalle en la sección de análisis. Los valores y la gestión de estos riesgos se han reflejado en la implementación de una forma de organización dinámica, adaptable y orientada a prioridades. Por todo lo anterior, las etapas de análisis, diseño e implementación no tienen unos límites bien definidos. Por ejemplo, el análisis de algunos riesgos ha requerido el diseño e implementación de pruebas de concepto y prototipos.

En la tercera etapa se ha validado el prototipo por expertos multidisciplinares, representando diferentes roles y aportando puntos de vista complementarios, y considerando tanto aspectos técnicos como educativos. Estos expertos también han aportado a esta validación la apreciación de usuarios finales, ya sean educadores o niños, estos últimos a través del ojo del experto. En esta validación no se ha utilizado la herramienta en el aula o con una finalidad puramente curricular. Para facilitar la tarea de evaluación, se ha diseñado un formulario estandarizado para todos los roles, con preguntas relevantes tanto sobre el software en sí, como sobre su uso educativo y la percepción de los usuarios finales.

En la última etapa se han comparado los resultados obtenidos con los deseados, se han analizado las carencias detectadas y se han establecido las posibles vías de desarrollo futuro.

La documentación del TFM se ha realizado paralelamente a la ejecución de estas etapas.

1.3. Estructura del documento

Este documento se divide en los siguientes capítulos:

6 Manifiesto del desarrollo ágil, http://www.agilemanifesto.org

1. Introducción. Este capítulo introductorio.
2. Estado del arte. En este capítulo se ha presentado una introducción a distintos conceptos de programación de los que se mencionan en ese y otros capítulos; se han analizado las herramientas existentes y posibles tendencias de la programación; y se han identificado las necesidades y características de la herramienta a diseñar.
3. Análisis del software. En este capítulo se ha realizado el análisis del software a desarrollar explicando los objetivos y requisitos del proyecto y cómo se trasladan estos a requisitos software.
4. Diseño. En este capítulo se han detallado decisiones de diseño y se ha descrito la estructura del software creado.
5. Implementación y pruebas. En este capítulo se ha aportado detalles sobre la implementación, se ha descrito en términos generales y de funcionamiento el software desarrollado y se ha explicado el procedimiento de pruebas usado en el desarrollo.
6. Validación. En este capítulo se ha descrito el proceso seguido para la validación del software y los resultados del mismo.
7. Conclusiones. En este capítulo se han discutido los resultados obtenidos y se han planteado las líneas futuras.

I. En este anexo se ha recogido el cuestionario de validación del software y sus resultados.

2. Estado del arte

En este capítulo se recoge el estado del arte relativo a las herramientas y los métodos aplicables a la enseñanza de la programación, incluyendo los antecedentes existentes, otras herramientas similares con la misma finalidad y los enfoques metodológicos.

2.1. Tecnologías y conceptos de programación

En este apartado se presentan algunos de los conceptos sobre programación más relevantes para el trabajo realizado, contextualizando los mismos en el uso educativo y profesional.

2.1.1. Programación visual y código fuente

Los primeros lenguajes de programación fueron construidos en base a lenguajes formales utilizando un juego predefinido de instrucciones (un léxico) y unas reglas de composición (una sintaxis). La facilidad para el tratamiento automático de texto ha hecho que esta forma de programar se mantenga hoy día en los lenguajes más habituales: Java, C#, JavaScript, Delphi, Ruby, etc. Un ejemplo de código se puede ver en la ilustración 2.1.

La programación visual se puede describir como aquella en que se usa un conjunto de objetos gráficos (nodos u objetos) que se relacionan entre sí (de forma explícita o implícita) para determinar la interacción entre los elementos, el flujo de datos o de programa, etc. (Cox, 2008). La programación puramente visual se da en dominios muy concretos, como por ejemplo Simulink[7], para la definición de modelos de simulación de sistemas físicos (ilustración 2.2); LabVIEW[8], para diseño de sistemas de adquisición de datos e instrumentación, ambos usados en ingeniería; en dominios muy próximos al hardware, como programación con puertas lógicas; etc. En un sentido amplio, también se suele considerar las hojas de cálculo como herramientas de programación visual en tanto y en cuanto se establecen relaciones espaciales entre los datos.

La programación visual recibió una enorme atención tanto académica como industrial en la década de 1990, y las posibilidades y limitaciones de la misma se documentaron ampliamente (Myers, 1990). Ya se destacaban las posibilidades educativas pero también las dificultades de los enfoques visuales en la programación de propósito general.

7 MathWorks Simulink, http://www.mathworks.es/products/simulink/index.html
8 National Instruments LabVIEW, http://www.ni.com/labview

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

Ilustración 2.1: Programa en Fortran (Wikipedia)

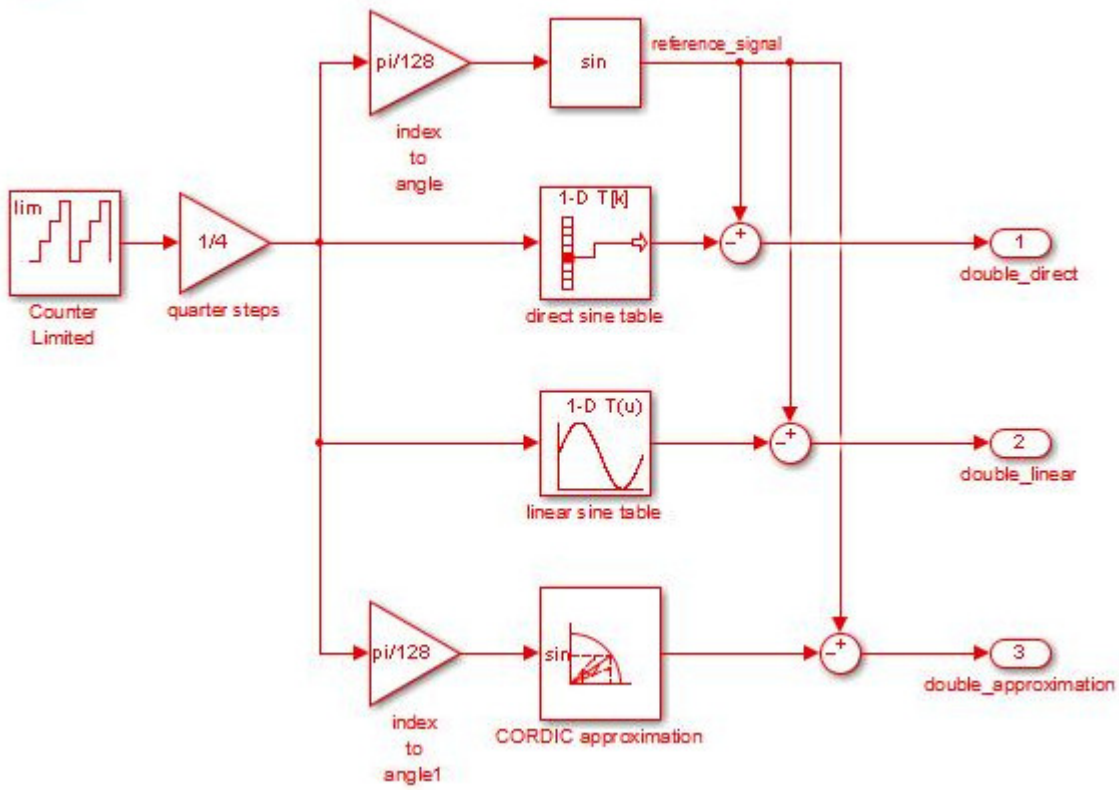


Ilustración 2.2: Programa de Simulink (Copyright 1990-2013 The MathWorks, Inc.)

Por otro lado, son muy frecuentes los entornos que utilizan una parte visual para el desarrollo de interfaces de usuario, prototipado rápido de aplicaciones y otras funciones muy específicas. Dependiendo del lenguaje y el dominio de uso se puede encontrar un balance hacia un extremo u otro. Por ejemplo, en los lenguajes más de propósito general como Java y C# existen potentes herramientas de desarrollo de interfaces de usuario y prototipado rápido. Pero estos entornos no aportan funciones nuevas (aunque pueden aumentar enormemente la productividad) y son totalmente opcionales. En el otro extremo, se pueden destacar ejemplos como herramientas de desarrollo de juegos que, como GameMaker[9], son visuales y pueden hacerse proyectos completos sin escribir una línea de código, pero disponen de un lenguaje de *scripting* creado *ad hoc* para personalizar los juegos y añadir características.

Distinguimos aquí entre programación visual y lo que podríamos llamar «asistentes visuales de sintaxis». Programación visual es lo que se hace al diseñar el flujo de un programa mediante bloques unidos mediante líneas. Un asistente visual de sintaxis es lo que hace Scratch (ilustración 2.3) o Blockly: componer código fuente arrastrando bloques de diferentes formas que solo encajan con los contiguos cuando forman una sintaxis formalmente correcta. En estos casos, transcribir el programa "visual" a uno textual es prácticamente trivial.

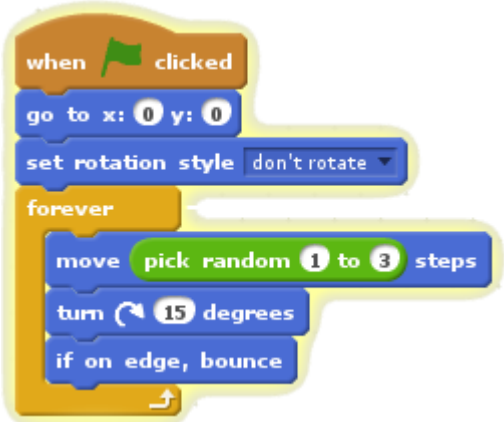


Ilustración 2.3: Programa en Scratch (scratch.mit.edu)

En el entorno educativo, las dificultades con el uso de la sintaxis se resuelven por dos caminos (Kelleher & Pausch, 2005), o bien simplificando la codificación de los programas o buscando alternativas a la escritura. Un ejemplo de lo primero es la sintaxis sencilla del lenguaje BASIC, acrónimo de Beginner's All-purpose Symbolic Instruction Code, uno de los primeros lenguajes de programación con un foco en la facilidad de uso (Kemeny & Kurtz, 1985), o los entornos que restringen la escritura de código al válido. Una alternativa a la escritura es proporcionar objetos predefinidos con los que construir los programas, como Scratch o los lenguajes visuales, o mediante la manipulación de las propiedades de objetos físicos simulados.

9 GameMaker Studio de YoYo Games, https://www.yoyogames.com/studio

En resumen, una forma de programación puede ser más aplicable que otra. Se presentan las características más importantes de la programación en un lenguaje formal (Kelleher & Pausch, 2005; Repenning, 2011; Schwartz, Morrison, & Witus, 2007):

- Más próximo a la programación «real».
- Foco en la sintaxis, incluso cuando hay un asistente visual o se *diseña* un lenguaje muy sencillo. El programa debe ser leído línea a línea y decodificado por el programador desde un lenguaje formal.
- Foco en los algoritmos.

Las principales características de la programación visual son:

- Foco en la estructura del programa y los datos.
- Más cercano al aprendizaje *visual*.
- Posiblemente, es más fácil hacer foco en los conceptos que se desean transmitir, en la *gran imagen*, que en los detalles (efecto *Lorem Ipsum*).
- El desarrollo de algoritmos detallados es muy complejo y farragoso. Este es el principal motivo por el cual casi todos los lenguajes de propósito general y uso profesional no son visuales.

2.1.2. Compilación e interpretación

En programación se distingue entre lenguajes compilados y lenguajes interpretados o de *script* (Pratt & Zelkowitz, 2000). Los lenguajes compilados son aquellos que no pueden ejecutarse directamente, sino que han de transformarse en un fichero «ejecutable» para poder funcionar. Los lenguajes interpretados se ejecutan en un entorno especial que lee cada instrucción del programa una a una y las ejecuta.

Aunque formalmente no hay distinción entre unos y otros (un lenguaje compilado puede ejecutarse de forma interpretada y viceversa), en la práctica hay diferencias: el entorno de ejecución de los *programas* compilados es el propio sistema operativo y por lo tanto se ejecuta a un nivel más bajo, inaccesible y opaco que los *programas* interpretados, cuyo entorno de ejecución es otro programa, como un intérprete de BASIC, de línea de comandos, un navegador o un entorno de desarrollo integrado.

2.1.3. Paradigmas de programación

Otra clasificación de los lenguajes de programación está en la forma en que se ejecutan e interpretan. Pueden distinguirse, entre otros, entre lenguajes imperativos y declarativos.

Los lenguajes imperativos son aquellos en los que su léxico define órdenes e instrucciones concretas, y la sintaxis define entre otras cosas el flujo de programa, es decir, qué orden se ejecuta después de la primera. Los lenguajes declarativos son aquellos en los que el léxico y la sintaxis describen el problema, las reglas y los datos, y la solución se obtiene aplicando reglas de inferencia entre los primeros (Pratt & Zelkowitz, 2000).

Además de estos, hay otros (sub)paradigmas que añaden facilidades y expresividad a los lenguajes:

- Programación estructurada: define estructuras de control del programa que permiten repetir o condicionar la ejecución de ciertas partes, es decir, bucles y condicionales.
- Programación procedural: distingue entre programas, subprogramas (métodos y funciones, subrutinas, etc.), que pueden ser invocadas desde otras partes del código.
- Programación orientada a objetos: define estructuras de datos y código como cajas negras, en las que el interior es desconocido e inaccesible salvo a través de las interfaces proporcionadas por el objeto.
- Programación demostrativa: el programa se genera extrayendo información de las operaciones que un usuario realiza manualmente sobre unos datos de entrada, y generalizándola (Cypher, 2010; James Lin, Wong, Nichols, Cypher, & Lau, 2009).

2.1.4. Programación de usuario final

El salto tanto tecnológico como filosófico más importante en los últimos años en el mundo de la informática, es el de la web 2.0, donde el usuario final se vuelve protagonista de los contenidos, como usuario y como creador o manipulador de los mismos. El usuario final escribe blogs, graba y edita vídeos, utiliza agregadores de contenido de diversas fuentes, comenta y reenvía noticias que considera de su interés o del de su comunidad, etc. Cabe esperar, por tanto, que el usuario se involucre más en personalizar el tratamiento automatizado de toda esa información. El término *programación de usuario final* lleva utilizándose desde hace más de 20 años para describir el trabajo de usuarios con las funciones de hoja de cálculo, paquetes estadísticos o programas de diseño asistido por

computador en áreas determinadas, y discutir las dificultades existentes para el salto hacia fuera de esos dominios concretos, y cómo enfrentarlas (Grinter, 1994; Hale, Solomonides, & Beeson, 2012), usando lenguajes de programación y herramientas específicos de la tarea o dominio, entornos de desarrollo visuales y prácticas de trabajo colaborativo.

Hale recomienda el uso de metodologías y herramientas que permitan el uso de herramientas visuales de alto nivel donde la estructura del software esté directamente reflejada en la visualización. Así se consigue llevar la comprensión del software a otro nivel, no solo por el experto del dominio sino por otros colaboradores como expertos en programación, y facilitar la mantenibilidad, extensibilidad y compartición de información. Tal y como Hale recoge, la programación visual usando técnicas como arrastrar y soltar en lugar de escribir código imposibilita cometer errores sintácticos y permite concentrarse en la semántica (Repenning, 2007), especialmente si tal semántica se ve reflejada en estructuras de programa con forma de grafos, en la misma manera que XML, RDF (*Resource Description Framework*) y OWL (*Web Ontology Language*), que describen semánticamente datos y las relaciones entre ellos (Rosson, 2007). Además, una forma de programación donde la semántica es protagonista y se abstrae el lenguaje de programación y los detalles de la implementación que se ejecutan es más fácilmente mantenible y ampliable.

La programación de *mashups* es un ejemplo de programación específica a dominios concretos por expertos en el dominio. Los mashups se definen como agregaciones de contenido con tratamiento de datos de diversas fuentes. Un ejemplo clásico es una búsqueda y comparación de un producto en diversas tiendas online y físicas: varios buscadores proporcionan información como coste del producto, del envío, localización, etc. Los datos de las búsquedas se filtran, eliminando las que están lejos, las que tienen peores valoraciones por la comunidad, las de peor tiempo de entrega, etc. Las que quedan se pintan en un mapa con etiquetas coloreadas por su rango de precio, incluyendo costes de desplazamiento o envío. La programación de estas herramientas por usuarios finales puede ser muy compleja si ha de realizarse en un lenguaje de programación que entiende solo de cadenas de caracteres y *sockets* de datos. Por ello, se han ido creando herramientas visuales que permiten desarrollar mashups en ciertos dominios sin conocimientos avanzados de programación, como Vegemite (James Lin et al., 2009) o Mashroom (Wang, Yang, & Han, 2009), que usan tablas, o diagramas de flujo (Cheeseman, Nguyen, & Osecki, 2009).

2.1.5. Programación concurrente

En los últimos años estamos asistiendo a cambios también importantes en la forma en que se programa. Uno de ellos, relativo a la propia naturaleza de la programación, es el salto a la programación concurrente o multitarea, al acabarse la época dorada de crecimiento exponencial en la velocidad de reloj los microprocesadores (Sutter, 2005). Si en 2001 aparecieron los primeros microprocesadores a 2 GHz, siguiendo los ritmos que se llevaban, los de 10 GHz se esperaban para 2005, cosa que sabemos no ocurrió (en 2014 los procesadores comerciales más rápidos siguen estancados por debajo de 4 GHz). La velocidad de procesamiento en la práctica aumenta mediante otras técnicas: caché, *hyperthreading*, procesadores multinúcleo, etc. En esto, la programación imperativa, consistente en la ejecución serializada de instrucciones está en desventaja frente a la programación declarativa o funcional o incluso orientada a eventos, que se considera *implícitamente* concurrente o paralelizable. Incluso si existen construcciones nativas para incorporar la concurrencia a los lenguajes «habituales» como Java, C++ (desde la revisión estándar C11 o ISO/IEC 9899:2011) o C#, estas son de más bajo nivel y por construcción más complicadas de usar en el diseño de programas concurrentes (Erb, 2012). Por estos y otros motivos, se espera en el futuro una mayor relevancia de lenguajes multiparadigma, que incorporan conceptos tanto de programación imperativa como declarativa (técnicamente, la mayoría lo son en un grado u otro, pero se trata de la integración de dichos conceptos de forma elegante y natural en los lenguajes y herramientas). Entre estos lenguajes cabría destacar JavaScript por su ubicuidad en la red y presencia en la familia de estándares HTML5. Por portabilidad y multiplataforma, los lenguajes interpretados (como JavaScript) y los ejecutados en máquinas virtuales (programas que se compilan para una plataforma «teórica» y son distribuidos en ese código intermedio a las plataformas finales en las que un entorno de ejecución finalmente los ejecuta, como Java y C#) adquirirán más protagonismo sobre los compilados (como C o C++).

2.2. Herramientas e iniciativas

Tanto en la literatura como en el mercado hay numerosas herramientas que ayudan al aprendizaje de la programación, ya sea para niños o para adultos. Algunas de ellas han sido validadas metodológicamente (de forma experimental o cuasiexperimental) en el ámbito educativo, como Scratch, Alice, Logo, Kodu, etc.

En esta sección se han analizado algunas de las más significativas, con el objetivo de encontrar patrones para clasificarlas y detectar carencias en las mismas y nichos de uso por explotar. Aunque cualquier lenguaje de programación y entorno es susceptible de ser usado de forma educativa, se ha hecho foco en las herramientas cuyo fin es el educativo y no el desempeño profesional. Igualmente, se puede distinguir entre aquellas herramientas orientadas a niños, a adultos, a profesionales o a un ámbito más general, de las cuales también se ha incluido algún caso. Por último, también se han señalado algunas iniciativas que exceden del ámbito reducido de una única herramienta software.

2.2.1. El lenguaje de programación Logo

Logo fue una de las herramientas pioneras en la enseñanza de la programación. La primera versión data de 1969 (The Logo Foundation, 2011). Con el lenguaje de programación de Logo, imperativo e interpretado, el programador daba instrucciones a un cursor, la *tortuga*, que se movía por la pantalla trazando gráficos. Tuvo un gran impacto y de este concepto surgieron muchas variaciones: toma de datos del «entorno de dibujo», juegos de instrucciones más complejos, salto a las tres dimensiones, etc. Hoy día existen versiones con distintas características para los nuevos sistemas operativos.

Básicamente, un intérprete de Logo posee dos siguientes elementos: un editor de texto para la codificación de las instrucciones y un área gráfica de dibujo, que en la mayoría de las versiones no es interactiva (ilustración 2.4).

Sin embargo, la existencia de Logo nunca se convirtió en una presencia generalizada como herramienta de enseñanza de la programación en las escuelas o en los hogares. En la literatura se apuntan las siguientes causas (Resnick et al. 2009):

- Muchos niños tenían problemas con el uso de la sintaxis, a pesar de que esta es probablemente más sencilla que la de muchos lenguajes de uso profesional.
- Desconexión entre el dominio (cosas que se pueden hacer con el lenguaje) de Logo con problemas e intereses del mundo real.
- A menudo, se introducía Logo en entornos donde nadie podía aportar ayuda o guía, ya fuera para resolver problemas con el uso o para dirigir la exploración.

En la actualidad, es difícil señalar un intérprete de Logo como el canónico o principal. La Fundación Logo se dedica solo a recoger todos los recursos disponibles sobre Logo,

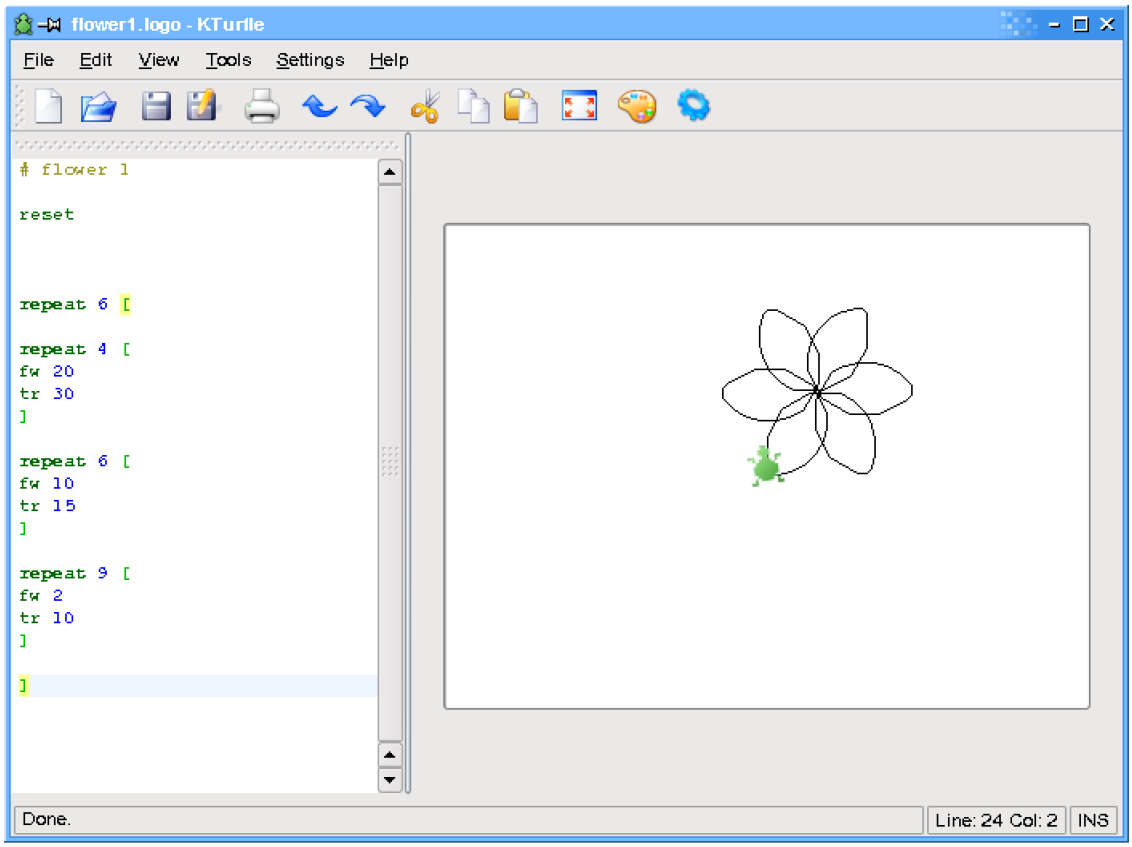


Ilustración 2.4: Programación con KTurtle, intérprete de Logo (Wikimedia Commons)

incluyendo los intérpretes desarrollados por terceras partes. Algunos intérpretes conocidos son UCBLogo, StarLogo, KTurtle, etc.

2.2.2. La organización *Code.org*

Code.org[10] es una organización sin ánimo de lucro en la que participan muchas grandes empresas estadounidenses para fomentar la inclusión en las escuelas del país de la programación y las ciencias de la computación en general.

Una de sus iniciativas de mayor impacto fue el *Hour of Code Challenge* (2013), en el que se publicaron cortos tutoriales interactivos destinados a niños de diferentes edades para que aprendieran a programar, dentro de la «Semana de la Educación en las Ciencias de la Computación».

Code.org usa los siguientes recursos educativos:

10 Code.org, http://code.org/

- El lenguaje de programación Blockly[11], un entorno de programación visual similar a Scratch.
- El lenguaje de programación Scratch (ver sección 2.2.4).
- Otros lenguajes de programación, como JavaScript y Python.
- Pequeñas aplicaciones o juegos Flash con un ámbito más reducido, como dar órdenes a un robot para que se mueva y ver cómo las ejecuta de forma no interactiva.
- Vídeos y abundante material para uso individual.

2.2.3. El portal Codecademy

Codecademy[12] es un portal que ofrece recursos para el aprendizaje autónomo de programación en seis lenguajes y entornos de uso profesional: Python, PHP, Ruby, JavaScript, jQuery (es una biblioteca que extiende funciones de JavaScript, en realidad no un lenguaje en sí mismo), HTML/CSS (es un lenguaje de marcado, no se programa en él sino que se define la estructura y aspecto de un documento, por lo que en este caso «programación» es un abuso del término). Incluye foros y otras herramientas colaborativas.

Codecademy participó junto con Code.org en la Semana de la Educación en las Ciencias de la Computación, pero está orientado hacia un público más generalista, tanto a profesionales como a aficionados que se inician en la programación en cualquier nivel.

El funcionamiento de los cursos en Codecademy consisten en microlecciones de mínima extensión donde se sigue la lección en un panel lateral, se introduce o modifica código en un editor de texto con algunas facilidades (numeración de líneas, sintaxis coloreada para facilitar la lectura e identificación del léxico y gramática del lenguaje) en la zona central, y se observan los resultados (solo texto) en una ventana o zona en otra parte de la pantalla.

Codecademy ha sido criticado desde varios frentes (Cooper, 2012), tanto desde el punto de vista educativo (falta de atención a los conceptos, foco innecesario en la sintaxis de los lenguajes, resolución mecánica de ejercicios) como de usabilidad (pobre información a errores frecuentes relacionados con la sintaxis, dificultad para el repaso), reconociendo también el potencial de la herramienta y en general la buena acogida recibida.

11 Google Blockly, https://code.google.com/p/blockly
12 Codecademy, http://www.codecademy.com/

2.2.4. El entorno Scratch

Scratch[13] es una de las herramienta de aprendizaje de programación para niños más reconocibles. Según sus creadores, "ayuda a los jóvenes a aprender a pensar creativamente, razonar sistemáticamente y trabajar colaborativamente". Apareció en 2006 y sigue siendo aplicada en el aprendizaje y usada en investigaciones sobre e-Learning (Olabe et al., 2014). Scratch es una aplicación web basada en Flash. La pantalla de trabajo (ilustración 2.5) tiene las siguientes partes:

- Un área de visualización, arriba a la izquierda.
- Un área de programa dividida en pestañas: en la primera se construye el programa mediante un asistente visual de sintaxis, arrastrando instrucciones representadas por bloques de diferentes colores y formas que encajan entre sí siguiendo las reglas de sintaxis a un «área de dibujo»; en la segunda y tercera se gestionan otros «recursos» del programa como ficheros gráficos o de audio.

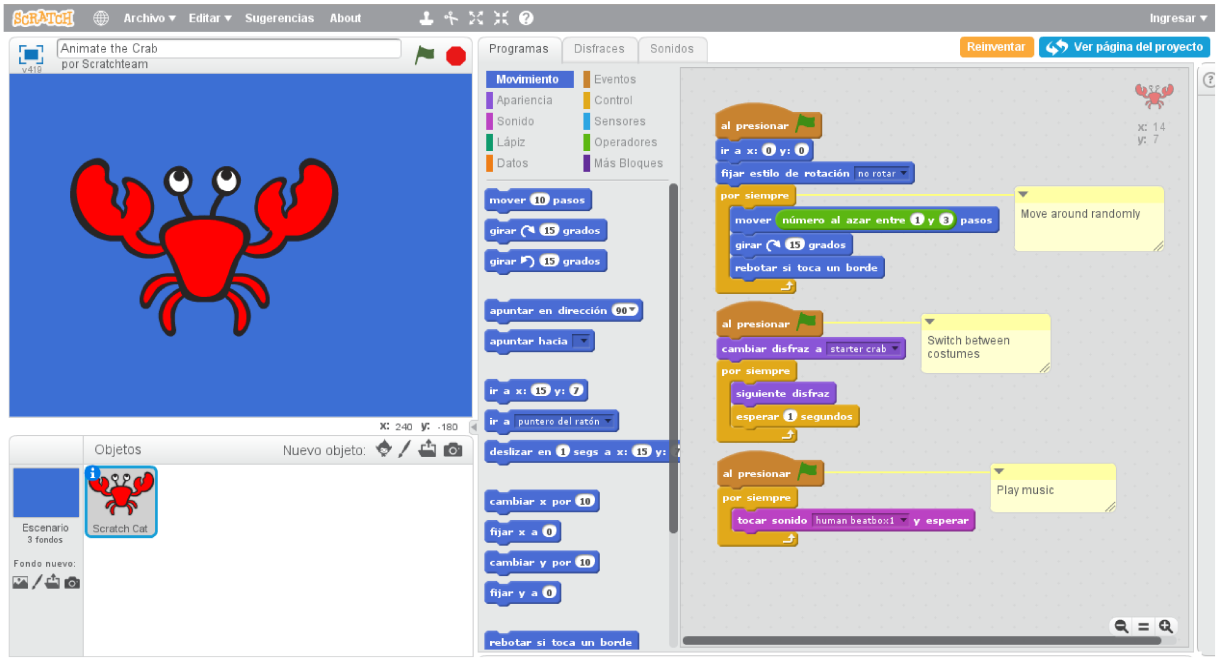


Ilustración 2.5: Escritorio de Scratch

Scratch fue creado con el triple objetivo (Resnick et al., 2009) de que fuera muy fácil iniciarse en él (*low floor*), que permita desarrollar proyectos complejos (*high ceiling*) y que permita muchos tipos de proyecto (*wide walls*). Los autores creen que gracias al asistente

13 Scratch http://scratch.mit.edu/about/

visual consiguen deshacerse de las complicaciones derivadas de la sintaxis de los lenguajes formales, uno de los problemas identificados a Logo. Con Scratch se pueden realizar proyectos de cierta complejidad: su sintaxis y biblioteca de funciones es bastante potente. El límite de Scratch se encuentra seguramente en cuántos bloques pueden desplegarse en un área de dibujo plana, sin «páginas», y seguir trabajando cómodamente. El último objetivo sirve para que personas con experiencias e intereses diferentes puedan encontrar una forma de expresarse y verse involucrados. Con Scratch se pueden hacer juegos, historias interactivas y no interactivas, tiene muchas funciones multimedia, los programas se pueden exportar a robots...

Scratch se basa en otros entornos educativos, entre otros LogoBlocks (Begel, 1996), que ofrece el mismo sistema de composición de programas mediante piezas. A su vez, y facilitado por la licencia MIT[14] de Scratch, han aparecido otros que siguen un sistema parecido: Blockly de Google, usado inicialmente en Code.org, y Hopscotch[15], ambos para aprendizaje por niños; Build Your Own Blocks (BYOB) o Snap[16], orientados a un público más adulto y con construcciones más avanzadas; Panther[17],basado en Scratch, que incorpora nuevas funciones como manipulación de páginas web...

2.2.5. Primo

Primo[18] es un juguete destinado a niños para que estos aprendan a programar. A diferencia de las otras que se han visto hasta ahora, no es una herramienta software, sino que es completamente «hardware». Primo tiene tres elementos: un «coche» controlado por radio, una caja de conexiones, que se puede programar y controla los movimientos del coche, y bloques con diferentes formas y colores (ilustración 2.6).

Los bloques realizan cuatro funciones: adelante, giro a la izquierda, giro a la derecha, llamada a «función», que se construye en el área marcada de la caja de conexiones. Después de conectar en la caja se ejecutan uno tras otro. Primo se podría considerar así como una herramienta de programación visual muy sencilla, con bloques que representan órdenes.

14 Licencia MIT de software libre del Instituto Tecnológico de Massachusetts, http://opensource.org/licenses/MIT
15 Hopscotch, https://www.gethopscotch.com
16 Snap! (Build Your Own Blocks), http://byob.berkeley.edu
17 Panther, basado en Scratch, http://pantherprogramming.weebly.com
18 Primo, http://www.primo.io

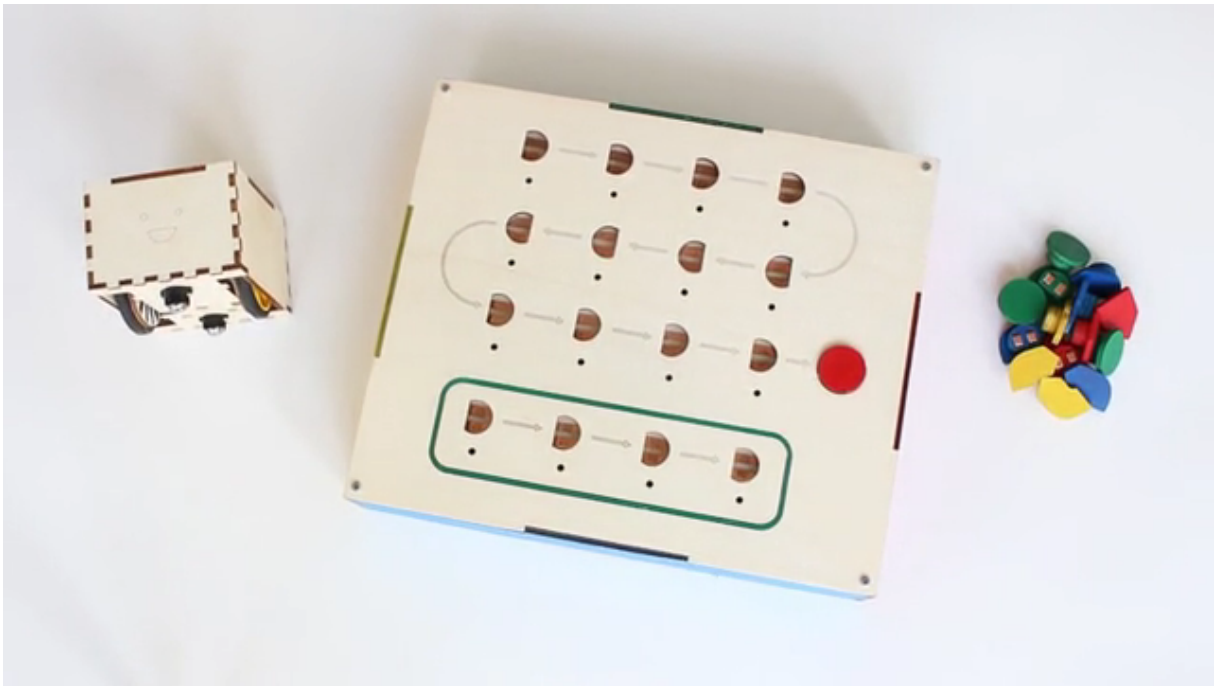


Ilustración 2.6: Elementos de Primo

Primo está orientado a edades comprendidas entre los cuatro y los siete años. No requiere ningún tipo de experiencia ni conocimientos previos, ni siquiera necesita saber leer. Obviamente, los conceptos que permite aprender son también bastante reducidos.

2.2.6. El entorno de programación Alice

Alice[19] es otro de los entornos de programación para el aprendizaje por niños «clásicos». Los autores de Scratch reconocen haberse inspirado en Alice para su desarrollo y también es frecuente encontrar referencias a Alice en el ámbito de la investigación en educación.

Alice es un entorno de programación rico en multimedia con el que se pueden crear historias y animaciones en mundos 3D. Además del entorno de programación, se ofrece una biblioteca de objetos y modelos de calidad profesional (cedida por Electronic Arts[20], una gran compañía de software de entretenimiento) para su uso en las historias. En la página web de Alice también se da acceso a material educativo tanto para alumnos como para profesores.

La forma de trabajar con Alice es mediante *drag & drop*. Alice está íntimamente ligado a los mundos 3D donde se desarrollan las historias. Una parte de «programar» con Alice es configurar el mundo 3D, añadiendo criaturas (objetos o instancias de clase), y la

19 Alice, http://www.alice.org
20 Electronic Arts, http://www.ea.com/about

otra es asignarles un comportamiento. En esto, al igual que Scratch, funciona con un mecanismo de asistente visual de sintaxis. Si en Scratch se trabajaba con una sintaxis muy simplificada y próxima al lenguaje natural, en Alice se programa de una forma bastante similar al lenguaje profesional Java, y hace bastante visibles conceptos como herencia, procedimientos y funciones y algunos aspectos de la sintaxis.

Un primer programa consistiría en definir el mundo (por ejemplo, dar textura y color al suelo y al cielo), añadir objetos (por ejemplo, un pingüino), y darle comportamiento. Entonces, el usuario selecciona de una lista el objeto sobre el que desea trabajar (el pingüino). En una pestaña pulsa un botón para añadir un método (comportamiento) y desde otra pestaña arrastra las definiciones de métodos (u otras operaciones o comentarios) ya existentes y las configura manipulando también de forma visual los argumentos de esas operaciones. Por último, añade la llamada a este procedimiento del usuario a un procedimiento inicial que se ejecuta siempre con el programa (ilustración 2.7).

Alice ha sido usado con éxito en jóvenes desde educación secundaria (Kelleher & Pausch, 2006) y hasta el primer año de educación universitaria (Moskal, Lurie, & Cooper, 2004) pese a algunas mostrar algunas carencias (Brown, 2008) en lo relativo a la transición de Alice a Java por la sintaxis y la abstracción de conceptos inherente a la parte visual. Otros problemas que se encontraron con el uso de Alice fue su dominio acotado a historias y animaciones, con limitaciones a la interacción y entrada y salida de datos.

2.2.7. El portal Kodu

Kodu Game Lab Community[21] es una iniciativa más reciente que Alice y Scratch que viene de Microsoft. Está diseñada para ordenadores y para X-Box, la videoconsola de Microsoft, y se puede usar con un mando de juegos, sin ratón o teclado. Además del entorno de desarrollo existe una comunidad para compartir las creaciones. No obstante, es software privativo y de pago en videoconsolas.

Su objetivo es el aprendizaje de la programación a través de la programación de juegos e historias y está orientado a niños a partir de los ocho años. La programación con Kodu consiste en usar objetos predefinidos o *bots* de una biblioteca que Kodu incorpora, distribuirlos en un terreno que es definido por usuario con herramientas básicas y darles comportamiento a través de menús con los que se asocian eventos a acciones como

21 Kodu Game Lab Community, http://www.kodugamelab.com

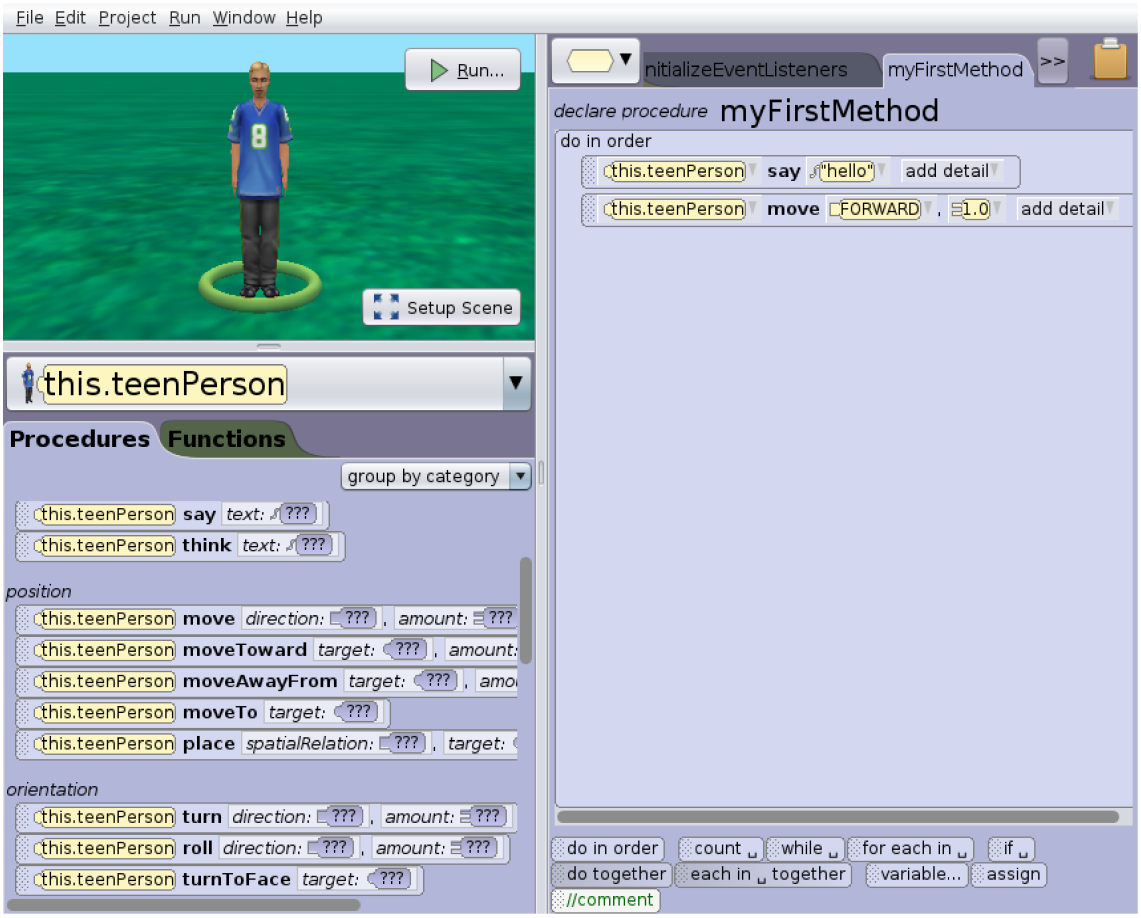


Ilustración 2.7: Trabajando con código en Alice

moverse o disparar (ilustración 2.8). La programación, así, es muy sencilla, pero permite mostrar conceptos como la programación orientada a eventos y lógica.

Kodu ha sido probado en el aula (Stolee & Fristoe, 2011) ha demostrado ser mejor herramienta para el aprendizaje que los lenguajes profesionales, mejorando la implicación de los alumnos (Fowler & Cusack, 2011).

2.2.8. El lenguaje y entorno Phrogram

Phrogram[22] es un entorno de programación y un lenguaje diseñado específicamente para usos educativos. Trata de disminuir la complejidad de uso mediante una sintaxis simplificada (en par a la de BASIC) y un entorno de programación más sencillo que los profesionales (ilustración 2.9).

22 Phrogram, http://phrogram.com

Ilustración 2.8: Programación con Kodu (Kodu Curriculum, Getting Started)

Sus autores consideraron que eliminar la parte de escribir código (y pelearse con la corrección de la sintaxis) era un enfoque erróneo (Schwartz et al., 2007). Trataron de crear una herramienta similar a las profesionales manteniendo las funciones y características de lenguajes de uso profesional como Visual Basic[23] (con Phrogram es posible crear programas multimedia con música y gráficos 2D y 3D como juegos), y basan en esto su valor como herramienta educativa.

A pesar de tener un enfoque diferente a otras herramientas, no ha tenido una gran difusión. Esto puede deberse a que su enfoque educativo (no deshacerse de la sintaxis) resultara al final ser poco apropiado y por su esquema de licencias (es software privativo y de pago). Como resultado, su primera versión data de 2006 y la última de 2008, aunque se ha mantenido la información y compatibilidad con las últimas versiones de Windows.

2.2.9. El entorno de programación Hackety Hack

Hackety Hack[24] es un entorno de programación muy sencillo para aprender a programar en el lenguaje de uso profesional Ruby. Ruby ya fue creado teniendo como

23 Microsoft Visual Basic, http://msdn.microsoft.com/en-us/vstudio/ms788229.aspx
24 Hackety Hack, http://hackety.com

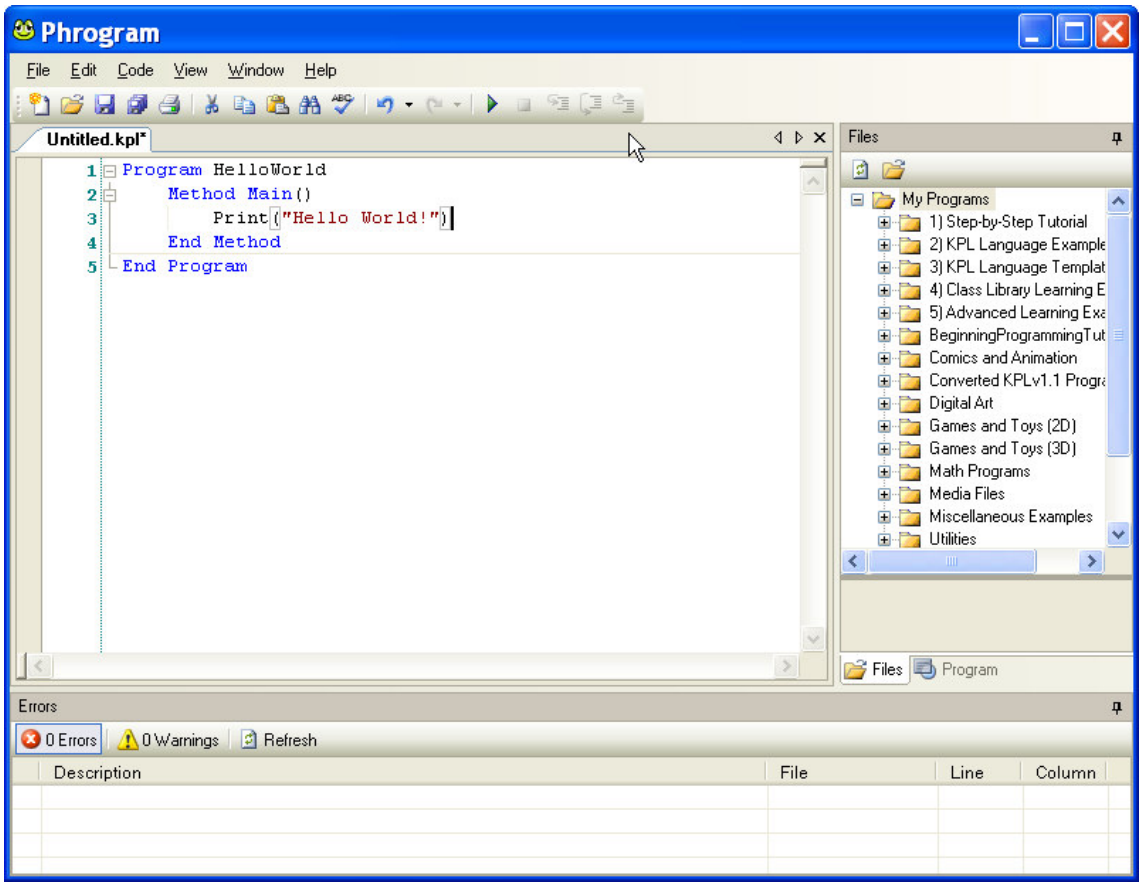


Ilustración 2.9: Entorno de programación Phrogram (Let's Program With Phrogram, tutorial de Jon Schwartz).

objetivo la simplicidad, y aún así es bastante potente y versátil, tomando conceptos de programación funcional e imperativa.

Hackety Hack no ha sido creado para niños sino para adultos, lo que se refleja en los materiales educativos que acompañan al programa: densos, muy basados en texto y con un lenguaje y uso del humor adulto. Por otro lado, los usuarios no necesitan conocimientos previos de programación para seguir las lecciones.

Hackety Hack destaca por su sencilla y agradable interfaz (ilustración 2.10). Con el material de soporte adecuado, podría llegar a ser apto para público no adulto, con las consideraciones oportunas acerca de escribir código en un lenguaje de programación real.

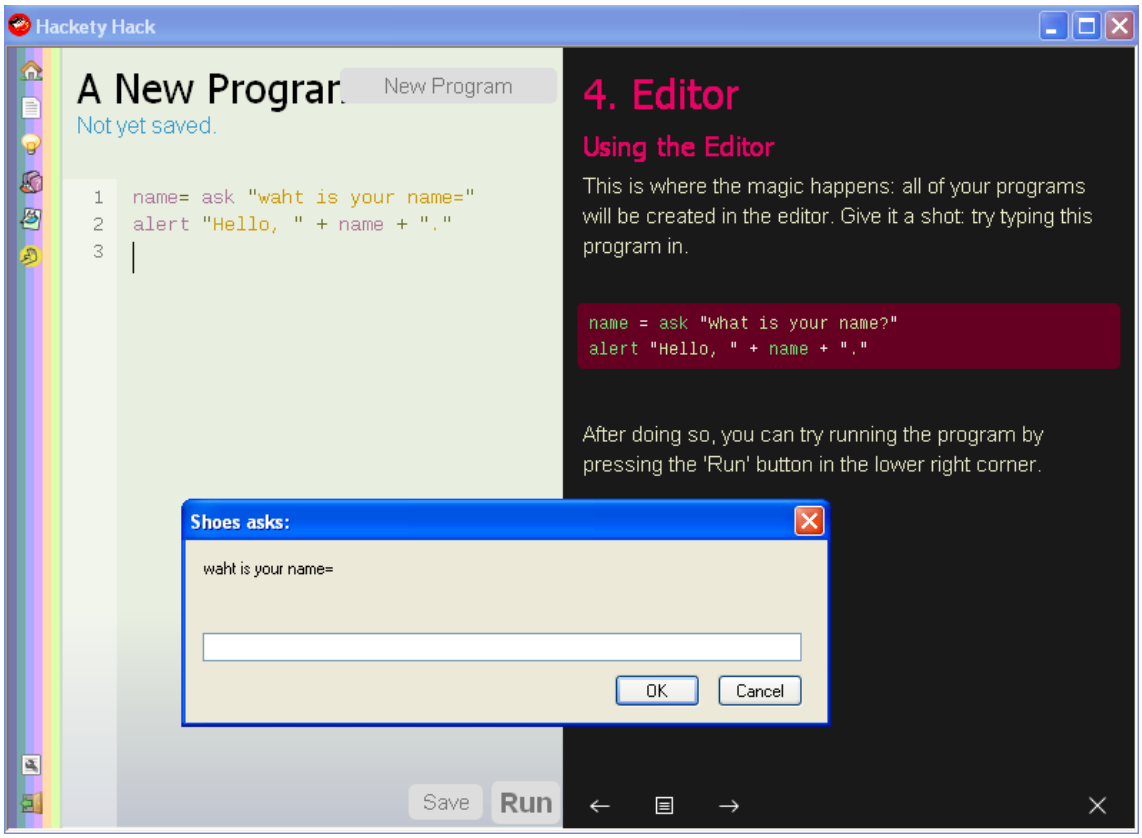


Ilustración 2.10: Programando con Hackety Hack

2.2.10. RoboMind

RoboMind[25] es una herramienta de programación inspirada en Logo que se caracteriza por ofrecer un mundo virtual en dos dimensiones para experimentar con la programación de robots sin necesidad de hardware (ilustración 2.11). Tal y como plantean los autores, los programas de Logo no eran suficientemente interactivos, y los robots de RoboMind pueden en ese mundo virtual interactuar con el entorno a través de sensores y actuadores virtuales.

RoboMind se plantea como una herramienta de introducción a la programación y la automatización con un rango de aplicación desde educación primaria hasta universitaria. Ofrece un lenguaje de programación con sintaxis sencilla, similar a Logo, en el que se pueden definir procedimientos, funciones recursivas y otros elementos avanzados.

25 Entorno de programación RoboMind, http://www.robomind.net

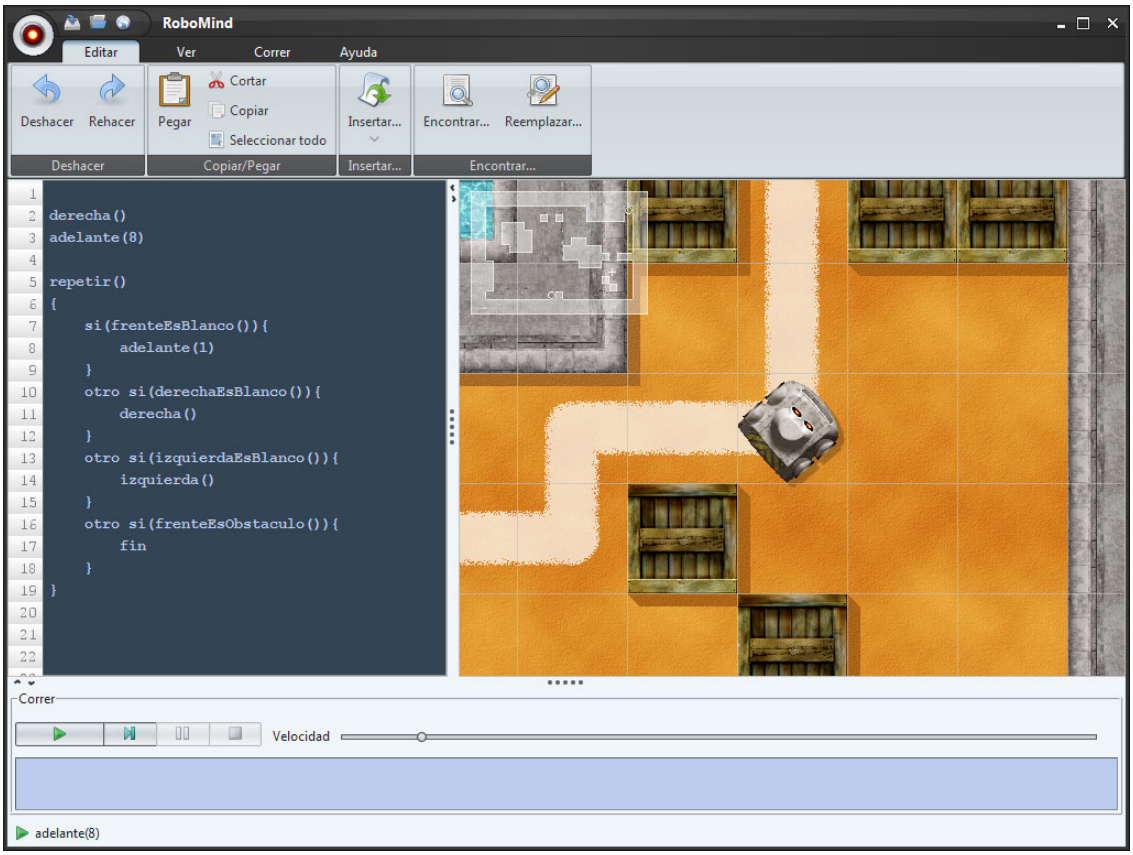


Ilustración 2.11: Programando con RoboMind (imagen de RoboMind.net)

2.2.11. Lego Mindstorms

Lego es un fabricante de juguetes de construcción y tiene una línea de juguetes avanzados con los que construir engranajes y máquinas sencillas. Con Lego Mindstorms[26] añade bloques programables que pueden controlar distintos tipos de motores y sensores, más un software para realizar la programación. Mindstorms ha sido desarrollado en colaboración con el MIT[27] y existe una versión para escuelas.

Mindstorms incorpora una herramienta de programación visual construida con LabVIEW aunque también es posible usar otros lenguajes de programación. Gracias a esto y a la variedad de sistemas sensores y motores ofrece una gran versatilidad como herramienta educativa, con experiencias de éxito en un rango tan dispar como estudiantes de entre 5 y 50 años (Erwin, Cyr, & Rogers, 2000)., para enseñar ciencia en escuela primaria, ingeniería e instrumentación a alumnos de universidad, etc.

26 Lego Mindstorms, http://mindstorms.lego.com
27 Instituto Tecnológico de Massachusetts, Estados Unidos de América, http://web.mit.edu/

2.2.12. Stencyl

Stencyl[28] es una popular herramienta de creación de videojuegos con gráficos 2D para ordenadores, móviles y web, orientada a aficionados y profesionales, ofreciendo herramientas de autoría y un entorno de programación, y cuenta con una activa comunidad de usuarios.

La programación en Stencyl puede ser tanto vía código fuente como visual a través de un asistente de sintaxis prácticamente idéntico al de Scratch (ilustración 2.12). Los desarrolladores de Stencyl son conscientes de la aplicación educativa de la herramienta y además de ofrecer licencias especiales a colegios cuentan también con kits educativos de dos semanas de duración.

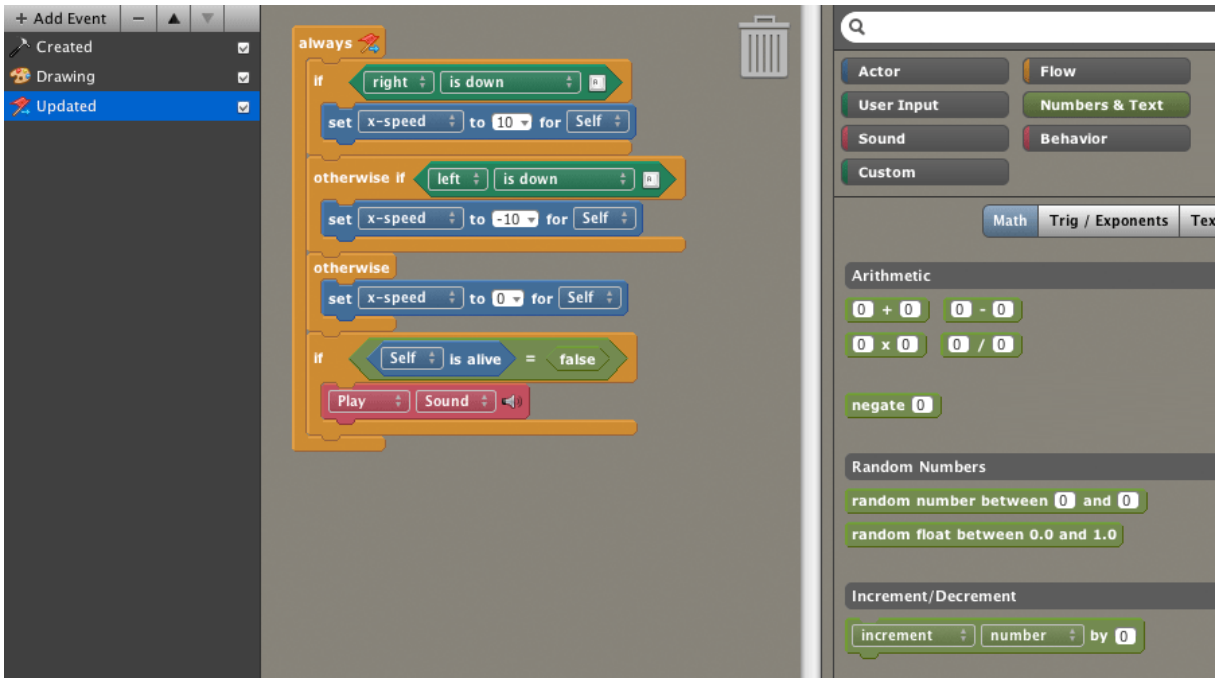


Ilustración 2.12: Programando con Stencyl 3.1 (imagen de Stencyl.com)

2.3. Clasificación de herramientas

Tras haber revisado algunas de las herramientas más significativas establecemos una serie de parámetros clasificatorios de las mismas.

2.3.1. Por dominio de aplicación

El dominio de aplicación, entendido como qué clase de programas es posible desarrollar con las herramientas, es característico de los entornos y lenguajes. Una forma de

28 Plataforma de creación de videojuegos Stencyl, http://www.stencyl.com

simplificar los lenguajes y herramientas es reducir sus funciones o adaptar la herramienta al desarrollo de un tipo concreto de trabajo. Así pues, hay un rango de especificidad, desde aquellos lenguajes y herramientas de propósito general, como la mayoría de los profesionales, hasta algunos de los que se han visto aquí, que solo pueden realizar (al menos de forma cómoda y directa) un determinado tipo de proyecto. Por otro lado, que el dominio sea amplio (*wide walls*) permite desarrollar la creatividad y expresividad del usuario y mantener el interés del mismo a lo largo de más tiempo. Llegar a un compromiso entre ambos extremos es un reto de las herramientas educativas.

Los lenguajes o herramientas de propósito general son aquellos que pueden ejecutar una gran variedad de funciones: cálculos, gráficos, entrada/salida, etc. La mayoría de los lenguajes de uso profesional son de propósito general: Java, C#, C++, Visual Basic, etc.

En el entorno educativo hay muchos dominios específicos: robótica, robótica virtual, historias y animaciones, videojuegos, gráficos de tortuga (*turtle graphics*, en referencia a los gráficos 2D hechos con la «tortuga» de Logo), etc.

2.3.2. Por forma de programación

Como ya se ha visto, se puede establecer una graduación entre programación «escrita» y programación puramente visual. La programación escrita con lenguajes formales siempre va a resultar más compleja que la programación puramente visual, debido a las dificultades prácticas y teóricas que impone la sintaxis del lenguaje. Sin embargo, la escritura de código hace más fácil la transición a los lenguajes de uso profesional, lo que será más relevante cuanta mayor sea la edad de los usuarios de la herramienta.

2.3.3. Relación con estilos de aprendizaje

Nótese que los diferentes modos de enfocar el aprendizaje a través de una herramienta tienen una proyección diferente sobre los términos usados en adaptación del aprendizaje. Quizá la proyección más obvia es la que se puede realizar sobre las inteligencias múltiples de Gardner (Gardner, 1999).

Como se ha visto, existe una graduación entre lenguajes textuales y lenguajes más visuales. Esto nos lleva a una correspondencia entre la inteligencia lingüístico-verbal y la inteligencia espacial.

Por otro lado, se ha mostrado que hay lenguajes imperativos, con foco en plasmar las órdenes que deben ejecutarse, y lenguajes declarativos, como los funcionales, con foco

en establecer las relaciones entre los objetos existentes. La correspondencia se hace aquí entre la inteligencia lingüístico-verbal y la inteligencia lógico-matemática.

El «círculo» puede cerrarse relacionando los lenguajes y herramientas con un foco en los algoritmos, no ya como secuencias de órdenes sino desde un punto de vista más general como operaciones o estados lógicos y matemáticos; y los lenguajes y herramientas visuales con foco en cómo se relacionan los objetos entre sí, marcando dichas relaciones el flujo de programa o de datos. Esta tercera correspondencia coloca los lenguajes entre las inteligencias lógico-matemática y espacial.

El resto de inteligencias de Gardner, musical, corporal-kinestésica, intrapersonal, interpersonal y naturalista, son más difíciles de traspasar al ámbito de la programación.

2.3.4. Otras consideraciones sobre las herramientas

Otras características a considerar en las clasificaciones de herramientas y lenguajes en el campo que nos atañe son:

- Público objetivo, es decir, las edades o fases escolares para las que la herramienta es más apropiada.
- Profundidad de los conceptos en los que instruye, a efectos comparativos. Se podría denominar *limitada* cuando hay conceptos avanzados no cubiertos, y *básica* cuando solo se cubren los conceptos más generales.
- Si es software libre o privativo.
- Plataformas en las que puede funcionar.
- Disponibilidad de la herramienta en múltiples idiomas.

2.3.5. Resumen de lenguajes y herramientas

En este apartado se resumen en varias tablas las características más relevantes de las herramientas analizadas.

Tabla 2.1: Herramientas e iniciativas analizadas

	Enlace	**Fechas de la primera y última versión**	
Logo	http://el.media.mit.edu/logo-foundation/	1967	[1]
Code.org	http://code.org/	2013	2014
Codecademy	http://www.codecademy.com/	2011	2014
Scratch	http://scratch.mit.edu/	2006 [2]	2013
Primo	http://www.primo.io	2013	2014
Alice	http://www.alice.org	1998	2014
Kodu	http://www.kodugamelab.com	2009	2014
Phrogram	http://phrogram.com	2005	2012
Hackety Hack	http://hackety.com	2009	2010
RoboMind	http://www.robomind.net	2005	2014
Lego Mindstorms	http://mindstorms.lego.com	1994	2013
Stencyl	http://www.stencyl.com	2011	2014
Otras menciones			
UCBLogo	http://www.cs.berkeley.edu/~bh/logo.html	1992	2009
StarLogo	http://education.mit.edu/starlogo/	2008	2011
Kturtle	http://edu.kde.org/kturtle/	2004	2014
Blockly	https://code.google.com/p/blockly	2012	2014
Snap	http://byob.berkeley.edu	2011	2014
Panther	http://pantherprogramming.weebly.com	2010	2011
Hopscotch	https://www.gethopscotch.com	2013	2014

[1] *Permanece como fundación recopilando todos los recursos sobre Logo.*

[2] *El origen del proyecto que dio lugar a Scratch data de 2003.*

Tabla 2.2: Dominio y público objetivo

	Dominio	**Público objetivo**
Logo	Gráficos de tortuga	Adolescentes – Adultos
Code.org	Propósito general	Escuela primaria y secundaria
Codecademy	Propósito general	Adultos
Scratch	Videojuegos, historias, animaciones, robótica	8 – 16 años [3]
Primo	Robótica	3 – 7 años
Alice	Historias, animaciones	Escuela secundaria
Kodu	Videojuegos	8 años en adelante
Phrogram	Videojuegos	7 años en adelante
Hackety Hack	Propósito general	Adultos
RoboMind	Robótica virtual	Escuela primaria y secundaria
Lego Mindstorms	Robótica	8 años en adelante
Stencyl	Videojuegos	Escuela primaria – Adultos
Otras menciones		
UCBLogo	Gráficos de tortuga	Adolescentes – Adultos
StarLogo	Gráficos de tortuga	Adolescentes – Adultos
Kturtle	Gráficos de tortuga	Adolescentes – Adultos
Blockly	Videojuegos, historias, animaciones	Niños – Adultos
Snap	Videojuegos, historias, animaciones	Adolescentes – Adultos
Panther	Videojuegos, historias, animaciones	Adolescentes – Adultos
Hopscotch	Videojuegos, historias, animaciones	Niños

[3] *También es usado por personas de todas las edades, incluyendo niños más pequeños con sus padres.*

Tabla 2.3: Forma de programación

	Programación	**Profundidad**
Logo	Sintaxis reducida	Básica
Code.org	Asistente visual de sintaxis / lenguajes de uso profesional	Limitada [4]
Codecademy	Lenguajes de uso profesional	Limitada [4]
Scratch	Asistente visual de sintaxis	Limitada
Primo	Con bloques físicos	Muy básica
Alice	Asistente visual de sintaxis	Limitada
Kodu	Visual	Básica
Phrogram	Sintaxis reducida	Limitada
Hackety Hack	Lenguajes de uso profesional	No limitada [5]
RoboMind	Sintaxis reducida	Limitada
Lego Mindstorms	Visual	Limitada
Stencyl	Asistente visual de sintaxis	Limitada
Otras menciones		
UCBLogo	Sintaxis reducida	Básica
StarLogo	Sintaxis reducida	Básica
Kturtle	Sintaxis reducida	Básica
Blockly	Asistente visual de sintaxis	Limitada
Snap	Asistente visual de sintaxis	Menos limitada que Scratch
Panther	Asistente visual de sintaxis	Menos limitada que Scratch
Hopscotch	Asistente visual de sintaxis	Más limitada que Scratch

[4] El formato de las lecciones y cursos limita el que sea utilizado de forma independiente para tratar otros conceptos más avanzados.

[5] El número de lecciones limita el aprendizaje guiado, pero el uso independiente es posible.

Tabla 2.4: Otras características del producto

	Licencia	**Plataforma de ejecución**	**Multiidioma**
Logo	---	---	---
Code.org	---	Web	Sí
Codecademy	---	Web	Sí
Scratch	Libre	Escritorio	Sí
Primo	Comercial	Hardware	---
Alice	Libre	Escritorio	No
Kodu	Comercial	Escritorio / Videoconsola	No
Phrogram	Comercial	Escritorio	No
Hackety Hack	Libre	Escritorio	No
RoboMind	Comercial	Escritorio	Sí
Lego Mindstorms	Comercial	Escritorio / Hardware	Sí
Stencyl	Comercial	Escritorio	Sí
Otras menciones			
UCBLogo	Libre	Escritorio	No
StarLogo	Libre	Escritorio	No
Kturtle	Libre	Escritorio	Sí
Blockly	Libre	Web	No
Snap	Libre	Web	Sí
Panther	Libre	Escritorio	No
Hopscotch	Comercial	Móviles	No

Tabla 2.5: Estilos de aprendizaje

	Estilos de aprendizaje principales
Logo y derivados	Lingüístico-verbal, lógico-matemática y espacial [6]
Code.org	Lingüístico-verbal y lógico-matemática
Codecademy	Lingüístico-verbal y lógico-matemática
Scratch y derivados	Equilibrado
Primo	Espacial
Alice	Lógico-matemática, lingüístico-verbal y espacial [6]
Kodu	Espacial y lógico-matemática
Phrogram	Lingüístico-verbal y lógico-matemática
Hackety Hack	Lingüístico-verbal y lógico-matemática
RoboMind	Equilibrado
Lego Mindstorms	Equilibrado
Stencyl	Equilibrado

[6] *Importancia de la componente espacial por el dominio de aplicación.*

2.4. Oportunidades identificadas

Al considerar el aprendizaje de la programación por niños con vistas a su uso en el mundo real y quizá profesional, hay que tener en cuenta cómo será la evolución de los lenguajes y herramientas de programación en el futuro. Quince años, el tiempo que puede transcurrir desde que un niño empieza a manejarse con un ordenador hasta que llega a una edad adulta y quizá de comienzo de actividad profesional relacionada, es un tiempo enorme en términos tecnológicos. De esta visión sobre las tendencias actuales pueden extraerse algunas ideas generales para el diseño de nuevas herramientas educativas para la programación:

- La programación visual y demostrativa son interesantes, por su facilidad para la comprensión y posibilidades futuras.
- La adscripción a un dominio concreto permite tanto simplificar el lenguaje y herramientas como involucrar al estudiante como usuario.
- El lenguaje JavaScript, que está muy extendido, tiene características muy buenas como que es interpretado y se ejecuta en entornos (navegadores web) disponibles ubicuamente, el multiparadigma, y habría que añadir otros puntos de los que no se ha hablado hasta ahora: que es un lenguaje débilmente tipado (que quiere decir que trata datos nativos, como números y cadenas, con mucha flexibilidad), que tiene una sintaxis poco estricta, y que está estandarizado (con colaboración con la industria por Ecma International[29]), formando parte de HTML5. Esto convierte a JavaScript a un lenguaje con prácticamente nulas amenazas de obsolescencia.

Como se ha comprobado, existe una tendencia al desarrollo de una programación visual y semántica que no se refleja en las herramientas encontradas, salvo en aproximaciones parciales como los asistentes visuales de sintaxis (Scratch, Alice), la programación-configuración a través de menús (Kodu), o la personalización y edición de «mundos» en 2D o 3D (Alice, Kodu, etc.).

Por otro lado, entre todas las herramientas existentes identificamos unas con dominios muy abiertos (como Scratch), otros muy cerrados en el dibujo (Logo y derivados), otros que solo sirven para crear historias y animaciones (Alice), otros para juegos (Kodu, Phrogram). Un dominio muy abierto puede ser una ventaja para llegar a un público general

29 Organización de estandarización Ecma International, http://www.ecma-international.org/

(*wide walls*), pero los dominios más restringidos no han impedido a otras herramientas conseguir su realización como herramienta educativa. Además, circunscribirse a un determinado dominio hace más sencillo crear las herramientas necesarias para atraer a un usuario, para facilitar los objetivos del mismo y para servir como vehículo de la enseñanza que se quiere ofrecer. Uno de los dominios que hemos visto que no están tratados con un mínimo de entidad y protagonismo es el musical.

En la discusión sobre los estilos de aprendizaje se ha establecido una asociación entre distintas formas de programación y las inteligencias más implicadas en ellas. La inteligencia lingüístico-verbal es la más habitual por la prominencia de código escrito y lenguajes formales, seguida de la lógico-matemática (en la algorítmica y la lógica inherente a la programación) y por último la visual (reflejada en la estructura visual de los programas, diagramas de flujo y de la interacción entre el usuario y los mundos 2D y 3D generados en los programas). De las restantes, por la relación entre la música y las matemáticas, y unido al aumento de capacidades multimedia en ordenadores, aplicaciones y web, la inteligencia musical es posiblemente fácil de incorporar al entorno de la programación.

Con todo esto, parece haber hueco para más herramientas, en particular, para una que integre una programación verdaderamente visual, más semántica que las existentes hasta ahora y menos algorítmica y verbal, aprovechándose de un dominio poco explotado como es el musical. Esta «fusión» se puede plasmar en una herramienta que facilite la creación, la manipulación y el análisis de la música, tocar instrumentos virtuales o cantar, que también use metáforas relacionadas con la música, etc. Y esto, en un entorno de programación visual muy simplificado que dé protagonismo a la estructura del programa y las relaciones entre sus componentes.

3. Análisis del software

El objetivo general de este TFM es **desarrollar una herramienta software para niños con la que introducir los conceptos de programación y facilitar el aprendizaje y la práctica de la programación**. Este objetivo se materializa en el desarrollo del software **APaM**, *Aprender Programación Haciendo Música*.

Se desglosa este objetivo general en otros más concretos de cara a comprender mejor el objetivo general del proyecto. Téngase en cuenta que al consistir este en un proyecto de desarrollo software, se han descompuesto algunos de estos objetivos específicos en requisitos software. Por lo tanto, estos objetivos se pueden dividir en dos grandes grupos, aquellos que han guiado a la necesidad de crear esta herramienta, y aquellos que dan lugar a requisitos software concretos. Los objetivos que guían a esta necesidad son:

- Describir las motivaciones para introducir la programación, una actividad hoy principalmente profesional y muy técnica, a niños.
- Analizar las herramientas existentes con objetivos similares: aprendizaje de programación en general y por niños en particular.
- Descubrir los enfoques aún noveles en este campo, ya sea en metodología, en dominio, etc.
- Diseñar un prototipo de herramienta basada en una metodología de aprendizaje constructivista que incluya la presentación progresiva de conceptos y la práctica en un dominio de operación que tenga la música como protagonista.
- Validar la herramienta desde el punto de vista educativo y tecnológico.

3.1. Requisitos software

Los objetivos que se han descompuesto en requisitos software son:

- Crear una herramienta software que permita *exponer* conceptos y técnicas usados en programación.
- Dotar a esta herramienta de la funcionalidad para *practicar* estos conceptos y técnicas, como un entorno de desarrollo integrado (IDE) convencional muy simplificado.

- Diseñar la interacción de esta herramienta para que pueda ser usada fácilmente por niños.
- Hacer esta herramienta accesible desde un ordenador de escritorio y desde una tableta, que son los dispositivos a los que más acceso puede tener un niño.
- Estilizar esta herramienta y sus componentes en un dominio musical.
- Diseñar e implementar un tutorial guiado que presente adecuadamente los conceptos.
- Elegir las herramientas de motivación adecuadas, como elementos gamificados (medallas y logros), desbloqueo progresivo de funciones, etc.
- Implementar en el tutorial guiado dichas herramientas de motivación.
- Verificar la usabilidad de APaM considerando su público objetivo.
- Verificar la aplicabilidad de APaM para el objetivo general propuesto.

3.2. Análisis de requisitos

En las metodologías ágiles el análisis de requisitos está orientado a establecer prioridades para las funcionalidades y capacidades de la aplicación, con el fin de poder incorporarlas según el valor para el cliente de cada función o según otras estrategias para minimizar riesgos de proyecto. Este análisis ha reflejado esta situación y se divide en los siguientes apartados:

1. Análisis de riesgos. Se han analizado los riesgos más importantes del proyecto y discutido las medidas tomadas para minimizarlos.
2. Historias de usuario. Las historias de usuario representan los requisitos de la aplicación expresados en un lenguaje sencillo.
3. Diagrama de arquitectura. El diagrama de arquitectura muestra las relaciones entre los componentes más significativos de la aplicación y su relación con elementos externos.

3.2.1. Análisis de riesgos

Los riesgos más importantes que se identificaron para este desarrollo software son:

- Viabilidad tecnológica de los objetivos planteados.

El diseño de una aplicación rica en interactividad (diseño de una compleja herramienta que dé una libertad análoga a la de un entorno de desarrollo integrado convencional) y recursos multimedia (por un lado, el dominio musical; por otro, el grafismo animado para niños; por último, los recursos interactivos usados en el tutorial) plantea exigencias importantes a las plataformas tecnológicas disponibles. Esto entra en conflicto con los requisitos de multiplataforma e independencia del dispositivo. Al elegir una plataforma determinada, pueden darse problemas para conseguir ciertos efectos después.

Para minimizar los riesgos relacionados con la viabilidad tecnológica, al principio del proyecto se realizaron varias pruebas de concepto y se investigaron bibliotecas y demostraciones tecnológicas existentes, con el fin de validar la disponibilidad de las funciones previstas y potenciales en aquel momento. Como ejemplo de esto, se compararon las funciones de generación de sonidos MIDI en JavaScript con la generación matemática de audio (a través de señales senoidales, cuadradas, etc.), con la grabación de multimedia a través del micrófono y tratamiento digital; y estas funciones con las disponibles en aplicaciones no web. Igualmente, durante el desarrollo, se priorizaron las funcionalidades más técnicas y complejas (funciones de IDE, interacción con la aplicación, habilitación del multimedia) frente a las del tutorial (tecnológicamente más simple).

- Incertidumbre debida a la elección de una tecnología de desarrollo determinada.

La elección de una tecnología determinada, como por ejemplo HTML5 frente a Java o C# además de sus consecuencias en la disponibilidad de funciones y de soporte multiplataforma para las funciones y bibliotecas que se quieren usar, impone también riesgos relacionados con la experiencia del desarrollador en tal tecnología. Distintos tipos de proyectos se realizan mejor en ciertas tecnologías, y lo que es fácil de hacer en una no tiene por qué serlo en otra. Del mismo modo, las herramientas de desarrollo no son las mismas y condicionan el rendimiento del desarrollo.

Los riesgos relacionados con esto se afrontaron principalmente mediante una búsqueda y evaluación de entornos de desarrollo adecuados a la tecnología elegida, que como se comenta en la sección correspondiente, fue HTML5 utilizando el framework jQuery. Los IDE de Java, C# y otros lenguajes más orientados a aplicaciones de escritorio, como Eclipse, QtCreator o Visual Studio, son excelentes para el desarrollo de ese tipo de aplicaciones por su elevada integración, pero para HTML5 y aplicaciones Web donde la aplicación se ejecuta fuera del entorno de desarrollo la elección de las herramientas es más

complicada. Por lo tanto, se evaluaron distintos entornos de edición HTML como BlueFish, Aptana y Eclipse, valorando la funciones de ayuda como API integrada, reconocimiento de sintaxis y autocompletar, estabilidad, etc. También se evaluaron distintas herramientas de soporte a desarrolladores en el navegador, como las integradas en Chrome y Firefox y plugins como Firebug. Finalmente se eligieron Firefox con Firebug y Eclipse con un plugin para facilitar el desarrollo con jQuery.

Igualmente, el desarrollador hizo su propia autoevaluación sobre sus capacidades con las distintas tecnologías que se plantearon usar, como Java, C#, JavaScript y HTML5.

- Incertidumbre en la planificación y gestión de tiempos.

Con la incertidumbre en tecnologías la gestión de tiempos y planificación es complicada, por la imposibilidad de prever tiempos y costes de desarrollo. Por ello, se buscó la comunicación frecuente con el tutor y permitirle la posibilidad de revisar el trabajo continuamente. Para ello, una vez que se decidió realizar una aplicación Web, se buscó un *hosting* donde poder desplegar la aplicación y tenerla actualizada manteniendo la privacidad del proyecto. Además, se implantó un sistema de gestión de versiones en el entorno de desarrollo para mantener el historial del desarrollo. También un sistema de gestión del tiempo consumido y en qué actividades para mantener un control del tiempo disponible.

El nivel de comunicación existente entre desarrollador y tutor no hizo necesario el uso de otras herramientas colaborativas de gestión, aunque se evaluaron soluciones como Producteev, BaseCamp y Assembla.

- Dificultad para evaluar el trabajo con niños a lo largo del desarrollo.

La aplicación no se pudo evaluar con niños hasta etapas tardías del desarrollo, debido por un lado a que las funciones de tutorial y ayuda guiada se dejaron para el final, y por otro al no tener hijos o alumnos en el rango de edad deseado.

- Conseguir un enfoque de programación visual exitoso.

Diseñar una programación puramente visual, completa desde el punto de vista semántico, es decir, capaz de crear todo tipo de programas, podría resultar en un procedimiento de programación demasiado complejo para el público objetivo. Por ello, se decidió realizar un compromiso entre los planteamientos de programación visual y no visual, realizando una programación auténticamente visual pero permitiendo, llegado un punto, introducir código fuente en un lenguaje formal. La programación visual se realiza en la forma

de un diagrama de flujo de datos con bloques conectados entre sí, complementada con bloques que admiten código fuente de verdad. Este enfoque es el usado en Simulink entre otros. En esta decisión se ha primado el conseguir el foco en los conceptos frente al foco en la sintaxis, pero permitiendo el uso de algoritmos complejos para «extender el techo» cuando se dominan los conceptos sencillos.

3.2.2. Historias de usuario

Las historias de usuario reflejan los requisitos de la aplicación desde el punto de vista y con el lenguaje del propio usuario. Los usuarios de esta aplicación son principalmente los niños pero también los educadores (profesores, tutores o padres) que deseen utilizar esta herramienta con sus alumnos. La historia de usuario representa, por tanto, el rol de usuario con el programa, qué es lo que quiere y cuál es el beneficio o resultado esperado.

La granularidad de las historias debe ser tal que permita su priorización (cada historia lleva una prioridad asociada, por su importancia para el proyecto), su negociación (poder tratar las historias individualmente y rechazarlas si se reevalúan o se descubren imposibles de llevar a cabo) y su estimación y planificación en tiempo. Para un proyecto de desarrollo que ha de ser llevado a cabo por una sola persona, las historias de usuario son muy desagregadas. En un escenario normal, una historia de usuario puede representar una cantidad de trabajo de unas 10 horas, pero para un proyecto de en torno a 250 horas (considerando el TFM completo, equivalente a 10 créditos), de las cuales una parte sustancial se dedican a la redacción de la memoria del trabajo, la granularidad habitual es insuficiente.

Por otro lado, la expresión de historias de usuario en el lenguaje de un niño puede ser insuficiente, por lo que se han expresado pensando en las necesidades del niño.

En la tabla 3.1 se desglosan las historias de usuario, junto con un indicador orientativo de su prioridad (A: mayor prioridad; C: menor prioridad):

Tabla 3.1: Historias de usuario

ID	**Historia de usuario**	**Prioridad**
	Requisitos hardware/software	
1	Como educador, quiero que se ejecute en navegador para conseguir portabilidad y multiplataforma.	A

ID	Historia de usuario	Prioridad
2	Como educador, quiero que funcione al menos en las versiones más extendidas de un navegador Firefox, Chrome, Explorer, etc.	C
3	Como educador, quiero que funcione en el sistema operativo Linux, Windows, Mac y Android.	C
4	Como usuario niño, quiero que se pueda ejecutar en un ordenador de escritorio, porque son frecuentes y es fácil disponer de uno.	B
5	Como usuario niño, quiero que se pueda ejecutar en una tableta, porque mi padre tiene una y es más cómoda que el ordenador.	B
6	Como educador, quiero que use las tecnologías HTML5 (JavaScript, CSS, etc.) y evite otras tecnologías como Java y Flash para mejorar la portabilidad.	A
7	Como usuario niño, quiero que se pueda ejecutar sin necesidad de registrarse en una página web o hacer login en alguna plataforma online por motivos de privacidad.	A
8	Como usuario niño, quiero que los datos se almacenen de forma que se minimice la interacción con el sistema operativo, su sistema de ficheros, etc., por facilidad de uso.	C
9	Como educador, quiero que funcione en equipos de potencia limitada y posiblemente antiguos, para poder usar los que tenemos en el colegio.	B
	Requisitos de usabilidad/accesibilidad	
10	Como usuario niño, quiero que funcione de la forma más parecida entre un ordenador y una tableta, o entre diferentes ordenadores.	B
11	Como usuario niño, quiero que sea fácil de usar (supere ciertos mínimos de usabilidad, medida en un test o valoración).	A
12	Como usuario niño, quiero que se controle mediante ratón o de forma táctil.	B
13	Como usuario niño, quiero que la interfaz sea muy sencilla, con el mínimo de controles.	B
	Forma de programar	
14	Como educador, quiero poder programar sin escribir una sola línea de código, de forma visual, con el ratón, para poder concentrarme en los conceptos más sencillos.	A

ID	Historia de usuario	Prioridad
15	Como educador, quiero poder programar también mediante líneas de código, para poder mostrar conceptos más complicados y cómo se almacenan y codifican los programas realmente.	A
16	Como educador, quiero poder programar en un lenguaje “común” y extendido, para facilitar la aplicación de lo aprendido al mundo real.	B
17	Como educador, quiero que con cada acción que modifique el programa se muestre o indique el equivalente en “código”.	B
18	Como educador, quiero que se programe mediante elementos de lo más alto nivel posible, como por ejemplo con programación orientada a eventos usando un mecanismo de señales y ranuras o *slots*, para concentrar la atención en los conceptos clave.	B
	Alcance de los conceptos a explicar	
19	Como educador, quiero que los conceptos de programación importantes se comenten al menos introductoriamente en la narrativa del software o tutorial, para que los estudiantes puedan familiarizarse con esos conceptos por su cuenta.	A
20	Como educador, quiero que la información sobre los conceptos de programación importantes se puedan ampliar al menos mediante enlaces externos fuera de la aplicación, como por ejemplo en la Wikipedia, para que los estudiantes puedan ampliar sus conocimientos por su cuenta.	A
21	Como educador, quiero que el **concepto de programa** esté visible desde el comienzo de la aplicación, aportando la metáfora necesaria.	A
22	Como educador, quiero poder hacer y mostrar un programa “Hola, Mundo” con un mensaje de texto en muy pocos pasos, para mostrar el concepto de programa.	A
23	Como educador, quiero poder hacer y mostrar un programa “Hola, Mundo” musical en muy pocos pasos, para mostrar el concepto de programa y en el dominio musical.	A
24	Como educador, quiero poder presentar **los conceptos de clase e instancia** (orientación a objetos) con los elementos constructivos básicos del programa, como cajas con un comportamiento predefinido, con una metáfora adecuada, como que cada elemento constructivo es un tipo de monigote o criatura.	A

ID	Historia de usuario	Prioridad
25	Como educador, quiero poder presentar **el concepto de encapsulación** (orientación a objetos): cada criatura hace unas cosas pero no la vemos por dentro, solo su interfaz externa.	B
26	Como educador, quiero poder presentar **el concepto de herencia** (orientación a objetos), añadiendo comportamiento a una criatura y crear una nueva clase a partir de ella, o al menos indicando que todas las criaturas son de un tipo "común".	C
27	Como educador, quiero poder presentar **el concepto de datos tipados**: una criatura contiene datos de distintos tipos, y solo puede manejar los del tipo adecuado.	B
28	Como educador, quiero poder presentar **el concepto de tipos básicos y complejos de datos**: con criaturas que usan tipos de datos básicos (como cadenas y números) y tipos de datos complejos (como notas musicales en una melodía).	B
29	Como educador, quiero poder presentar **el concepto de programación basada en eventos**: cada criatura emite señales y las recibe en *slots*, con una metáfora adecuada.	A
30	Como educador, quiero poder presentar **el concepto de entrada de datos**, con criaturas que permitan introducir datos en el programa.	A
31	Como educador, quiero poder presentar **el concepto de salida de datos**, con criaturas que permitan mostrar datos generados por el programa.	A
32	Como educador, quiero poder presentar **el concepto de condicional** (programación estructurada), pudiendo implementar condiciones con elementos constructivos de alto nivel (criaturas) o con código.	A
33	Como educador, quiero poder presentar **el concepto de bucle** (programación estructurada), pudiendo implementar bucles con elementos constructivos de alto nivel (criaturas) o con código.	A
34	Como educador, quiero poder presentar **el concepto de función o procedimiento y reusabilidad**, pudiendo reutilizar programas completos con otros elementos constructivos, o alternativamente enfocándolo a los *slots*.	C
	Elementos educativos	

ID	Historia de usuario	Prioridad
35	Como usuario niño, quiero que haya un **modo tutorial**, narrativo y con escenarios, para hacer un recorrido por los conceptos básicos.	A
36	Como usuario niño, quiero que haya un **modo "libre"**, donde pueda usar todas las criaturas, para practicar y hacer aprendizaje por descubrimiento.	A
37	Como usuario niño, quiero que se me guíe paso a paso y se me expliquen las operaciones con **animaciones en pantalla**, para comprenderlas mejor.	B
38	Como usuario niño, quiero que las animaciones y explicaciones con imágenes tengan narración de voz o al menos texto.	B
39	Como usuario niño, quiero una indicación relevante y acorde al dominio de mi progreso en el tutorial y con la herramienta mediante logros, medallas o títulos.	C
	Elementos de la interfaz	
40	Como educador, quiero que haya un área de trabajo que represente el programa, con una metáfora acorde al dominio, como un «escenario».	A
41	Como usuario niño, quiero que haya un área en la que pueda encontrar las criaturas que quiero usar en mi programa.	A
42	Como usuario niño quiero una interfaz que me permita guardar mis programas y recuperarlos en otro momento.	B
43	Como usuario niño quiero que las criaturas sean fáciles de distinguir y sean simpáticas.	B
44	Como usuario niño quiero que las conexiones entre señales y *slots* entre criaturas sean visibles en el programa.	B
45	Como usuario niño quiero poder borrar cosas (criaturas o conexiones) si me he equivocado o no me hacen falta.	B
46	Como usuario niño quiero poder deshacer el borrado si me he equivocado al borrar.	B
	Criaturas	
47	Como educador, quiero que los elementos constructivos estén representados mediante criaturas y que estén relacionados con el dominio y sigan una ambientación determinada, para dar coherencia al conjunto.	A

ID	Historia de usuario	Prioridad
48	Como educador, quiero que cada criatura, que representará una instancia de una clase, tenga unas señales y *slots* definidos que no se pueden modificar.	A
49	Como educador, quiero que cada criatura tenga unos datos editables que alteren el comportamiento de la criatura.	A
50	Como usuario niño, quiero poder añadir las criaturas al escenario o moverlas simplemente arrastrando con el ratón o el dedo.	A
51	Como usuario niño, quiero poder ver de forma ordenada las señales, *slots* y datos de las criaturas haciendo clic o tocando con el dedo sobre ellas o sobre menús emergentes o popups.	A
	Catálogo de criaturas	
52	Como usuario, quiero una criatura «reproductor» que toque una melodía, con señales, slots y datos para controlar la reproducción y saber la melodía que está tocando.	A
53	Como usuario, quiero una criatura «anotador» que pueda almacenar la melodía tocada desde una o varias fuentes distintas.	B
54	Como usuario, quiero una criatura «grupo» que me permita reutilizar programas hechos anteriormente en programas nuevos.	C
55	Como usuario, quiero una criatura «inicio» que me permita iniciar un programa con unos datos iniciales.	A
56	Como usuario, quiero una criatura «fin» que me permita terminar un programa con unos datos finales.	A
57	Como usuario, quiero una criatura «piano» que me permita introducir datos en el programa durante la ejecución del mismo en forma de notas musicales.	A
58	Como usuario, quiero una criatura «metrónomo» que envíe señales periódicamente a otras criaturas.	B
59	Como usuario, quiero una criatura «contador» que cuente señales recibidas y mande la cuenta.	B
60	Como usuario, quiero otras criaturas acordes al dominio para la manipulación de la música.	B

3.2.3. Diagrama de arquitectura

En el diagrama de arquitectura se pueden ver las tecnologías usadas, dónde se usan y dónde se almacenan los datos existentes (ilustración 3.1).

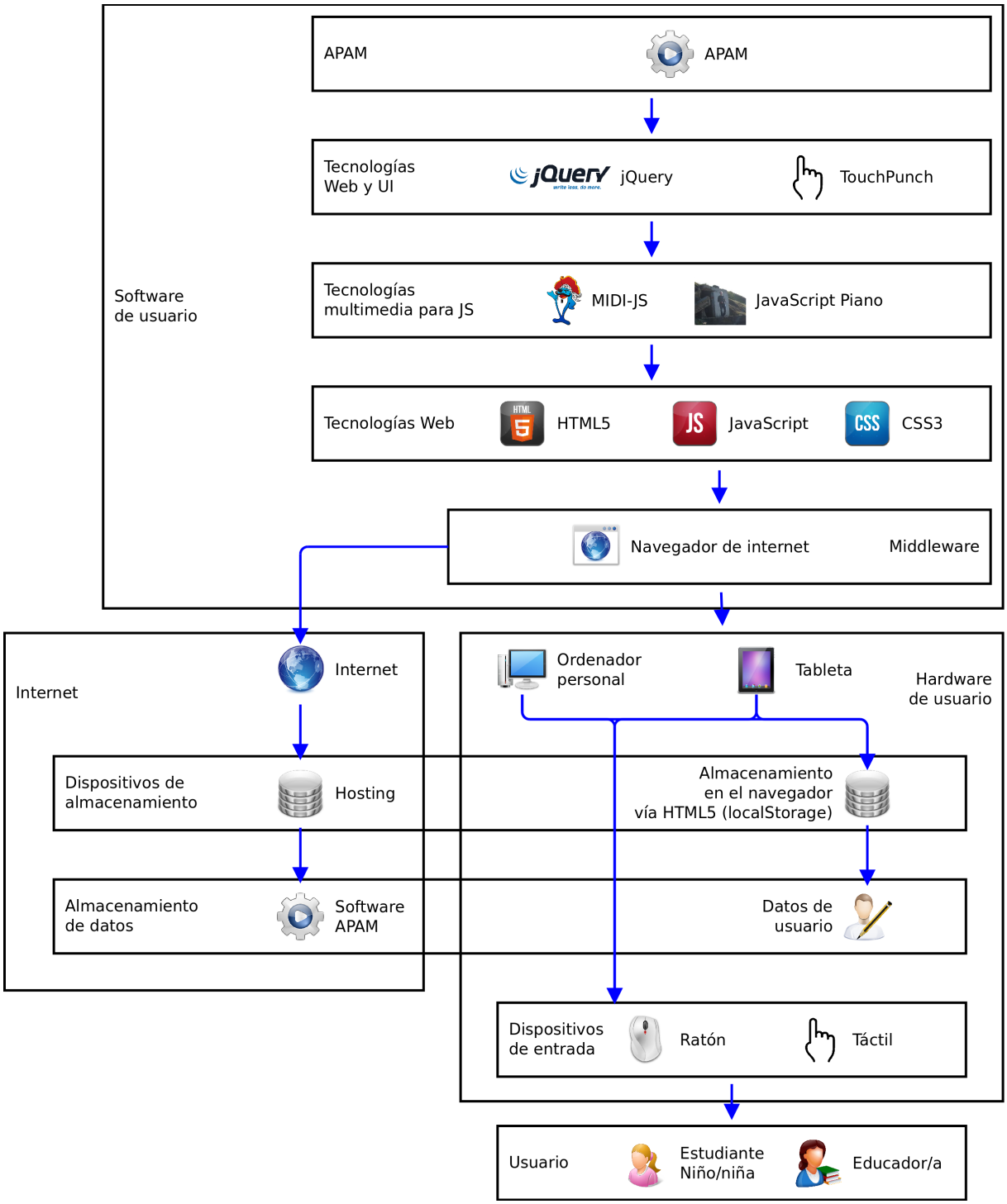


Ilustración 3.1: Diagrama de arquitectura

El software APaM se ejecuta íntegramente en el equipo del usuario, proporcionando el servidor únicamente el hosting de los datos. Los únicos datos de usuario que maneja APaM, como la personalización del usuario y los programas creados por él, se almacenan a través de las funcionalidades de HTML5 de almacenaje local, limitadas a 5 MiB por página web (5×1024×1024 octetos).

La interacción se realiza a través del ratón, si bien soporta también interfaces táctiles sin merma de funciones, gracias a la biblioteca TouchPunch. La interfaz de usuario está soportada por la biblioteca jQuery.

4. Diseño

En esta sección han mostrado los detalles más relevantes del proceso de diseño de APaM, incluyendo los siguientes apartados:

- Decisiones de diseño. Se han tratado en este apartado los condicionantes y motivaciones de las decisiones de diseño del software y contenidos más importantes.
- Diagramas de clases y objetos DOM. El software ha sido construido usando un lenguaje de programación orientado a objetos (JavaScript). En este apartado se ha descrito la estructura de clases que forma este software.

4.1. Decisiones de diseño

En todo proyecto software es necesario tomar decisiones continuamente acerca de tecnologías, diseños, patrones a usar, etc. Casi siempre estas decisiones no dependen de un único factor y no tienen soluciones únicas, por lo que la decisión adoptada suele servir a un compromiso entre los pros y contras de las diferentes alternativas. En este apartado se han descrito algunas de estas disyuntivas, la decisión tomada y el porqué de la misma.

4.1.1. Software de escritorio frente a aplicación Web 2.0

Una de las decisiones más cruciales es si desarrollar una aplicación de “escritorio”, que será ejecutada directamente por el sistema operativo del hardware (el entrecomillado es porque software de escritorio parece restringirse a las que se ejecutan en un ordenador de sobremesa o portátil, excluyendo a tabletas, móviles y otros dispositivo, lo que no se pretende), o una aplicación web sujeta a estándares web, que se ejecutará en un navegador.

- Ventajas de construir una aplicación de escritorio.
    - Integrada con el *look & feel* de la plataforma, ya sea Windows, iOS, Linux, etc., aunque esto solo se suele conseguir mediante la generación de una versión del programa para cada plataforma y a costa de mayor tiempo de desarrollo. Existen lenguajes y *frameworks* (conjuntos de bibliotecas) que permiten balancear la integración frente al tiempo de desarrollo como por ejemplo Java, C#/Mono o Qt.
    - Mejor rendimiento y más posibilidades en aplicaciones multimedia.

- Ventajas de construir una aplicación web.
    - La programación sujeta a estándares de aplicaciones web delega los problemas de compatibilidad multiplataforma a los desarrolladores de los navegadores. Aún es prerrogativa del desarrollador de la aplicación web el garantizar la compatibilidad con el mayor número posible de navegadores, pues estos no suelen cumplir los mismos conjuntos de especificaciones ni del mismo modo.
    - La ejecución en navegador garantiza el acceso homogéneo a interacción de usuario, funciones multimedia, etc. Como en el caso anterior, también es necesario estar atento a las diferencias entre navegadores.
    - Mucha mayor simplicidad en el despliegue. Una aplicación web siempre estará accesible desde una dirección URL sin necesidad de instalar software adicional. Es más, si la aplicación web satisface ciertos requisitos esta podrá ser desplegada incluso dentro de un recurso SCORM.
    - Es más fácil y homogéneo incorporar funciones de usabilidad y accesibilidad.

Una tercera opción podría ser una aplicación web que use plugins de navegador que amplíen las posibilidades multimedia, como por ejemplo Flash, pero al hacerlo se pierden las ventajas más interesantes en este contexto de las aplicaciones web, como la compatibilidad multiplataforma (por ejemplo, el plugin Flash de Adobe, casi un estándar para multimedia e interactividad en la web, no está disponible para iOS; Unity Web Player, no está disponible para Linux o Android).

En este caso, se ha dado más importancia a la disponibilidad multiplataforma y la facilidad en la distribución de la aplicación, por lo que se ha desarrollado como una aplicación web sujeta a los estándares de HTML5. HTML5 es mucho más que simplemente el lenguaje de marcado que eran las versiones antiguas de HTML. Es, más bien, una familia de tecnologías que incluyen JavaScript, varias API para recursos multimedia y gráficos 2D y vectoriales, geolocalización y otras muchas funciones.

4.1.2. Aplicación puramente cliente frente a aplicación cliente-servidor

Otra consideración a tener en cuenta es si la aplicación web se ejecutará completamente en cliente o si es dependiente de un entorno de ejecución en el servidor. Típicamente, las aplicaciones web que trabajan en base a cuentas de usuario y manejan

datos que se almacenan en remoto para ser accedidas desde cualquier lugar usan un *backend* Java, .NET o PHP. Esto añadiría una dependencia externa a la aplicación y mayor complicación a nivel de arquitectura, al requerir una comunicación continua entre cliente (la parte de APaM que se ejecuta en el navegador) y servidor. Esta dependencia también impediría que APaM se pudiera desplegar como un objeto de versiones antiguas de SCORM. Otra opción a considerar podría ser que solo sea necesaria la conexión externa para datos completamente opcionales.

La creación de bases de datos online y cuentas de usuario siempre tiene dos impactos a tener en cuenta: seguridad y privacidad.

La seguridad, es decir, preservar tanto los datos almacenados como el propio software garantizando su disponibilidad e integridad, requiere planteamientos específicos. Sería necesario poner medidas para garantizar la seguridad durante la comunicación de los datos enviados y recibidos (incluyendo datos sensibles como usuarios y contraseñas), para prevenir ataques de denegación de servicio maliciosos o accidentales (por un diseño incorrecto de la aplicación), etc. Otro problema, que es muy difícil de prevenir salvo mediante educación del usuario, son los ataques de *phishing*, consistentes en el robo de contraseñas y datos de usuario incitándoles a acceder a la aplicación atacada pero a través de un enlace que redirige un sitio falso copiado del original. Aunque los datos almacenados por APaM no sean sensibles y que estos queden comprometidos pueda ser poco importante, la compartición de contraseñas con distintos sitios es muy habitual y a partir del robo de contraseñas en un sitio es posible obtener acceso a cuentas con datos más críticos como correo electrónico o redes sociales. Dado que no se puede dar por sentada este tipo de educación en niños o incluso adolescentes, los aspectos relacionados con la seguridad pueden ser problemáticos para esta aplicación.

La privacidad, por supuesto, también es un detalle a tener muy en cuenta especialmente al diseñar un producto orientado a menores de edad. Incorporar funciones de tipo red social o simplemente de intercomunicación requiere planificar también mecanismos para proteger a los usuarios de comunicaciones dañinas o maliciosas, como gestión de contactos y bloqueos, detección de lenguaje ofensivo y bloqueo automático, gestión de «dominios de usuarios» como podría ser un grupo de alumnos de una clase, etc. Se podría plantear una instalación «por dominio» en un servidor no abierto a Internet sino solo a una subred de un colegio. Pero las ventajas que esto aporta frente a la creación de una aplicación web ejecutada íntegramente en el cliente, y así más fácil de integrar como un

objeto SCORM en otras plataformas de aprendizaje, quizá no justifiquen el esfuerzo de desarrollar estas funciones en una primera versión de APaM.

Por todo lo anterior, APaM se ejecuta íntegramente en el navegador del cliente, sin necesidad de conectarse a servidores remotos para intercambiar datos de usuario o sesión. La principal carencia que esto podría provocar (imposibilidad de almacenar los programas generados) se solventa dentro del mismo HTML5, que permite almacenar datos a través del navegador de forma solo accesible a la página web que sirve la aplicación. Al no necesitar almacenar datos sensibles como nombres o contraseñas se minimizan también los problemas de seguridad y privacidad. La comunicación con otros servidores puede limitarse a enlaces a páginas externas con información adicional como podría ser la Wikipedia.

4.1.3. Lenguaje de programación base

Esta decisión está bastante relacionada con las dos anteriores. De la anterior (programación visual frente a código fuente) se establece que hace falta un lenguaje de programación «base», que permita ir más allá de los conceptos iniciales y definir algoritmos detallados. De la primera (aplicación web frente a escritorio), se complica la aplicación de lenguajes *compilados* frente a *interpretados*, esto es, entre lenguajes que se transforman en un fichero binario (el programa) ejecutable directamente por el sistema operativo (como son la mayoría de lenguajes de propósito general usados en la programación profesional; en esta categoría estarían C# y Java) y lenguajes que son ejecutados interactivamente, línea a línea, por un entorno de ejecución (como JavaScript y otros lenguajes de *scripting*).

El uso de un lenguaje compilado requeriría ejecutar al menos una parte de la aplicación web en un servidor con capacidad para recibir el programa definido en el navegador del usuario, compilarlo, ejecutarlo y enviar la salida al navegador del cliente. El uso de un lenguaje interpretado requiere definir el lenguaje si se utiliza uno e implementar el intérprete. Si se usa JavaScript, que ya forma parte de HTML5, el mismo navegador sirve como entorno de ejecución.

Una opción a considerar dentro de los lenguajes compilados es el lenguaje ChucK[30], ya creado específicamente para el dominio musical. Este lenguaje también ha sido utilizado como vehículo del aprendizaje de la programación, si bien a adultos y a profesionales dentro del ramo de la música y artistas digitales. Aunque la coincidencia de dominios haría esta opción muy interesante, es necesario balancear lo que trata de aportar la presencia de un

30 Lenguaje de programación ChucK, http://chuck.cs.princeton.edu

lenguaje de programación de «base» a este proyecto. Y si como se ha señalado en el apartado anterior, es poder continuar el aprendizaje en el ámbito más detallado de los algoritmos particulares y de la aplicación al mundo real, JavaScript es preferible a ChucK.

Otras ventajas de JavaScript son las siguientes:

- Está amplísimamente extendido. Hay muchísimos recursos adicionales disponibles en Internet, destinados tanto a profesionales como a aficionados.
- No hace falta un servidor que actúe de compilador y ejecute el programa, con lo que es más fácil integrarlo con el resto de la plataforma.
- Puede interactuar con los elementos existentes en el programa con facilidad.
- Disminuye el tiempo de desarrollo y de creación de contenidos específicos para el aprendizaje del lenguaje.

En APaM se usa JavaScript como lenguaje de soporte.

4.1.4. Paradigma de programación

Se diseñó la forma de programar en APaM de forma visual dando protagonismo al flujo de programa y de datos, y a los componentes frente al código. Así, se decidió usar un paradigma basado en la orientación a eventos. En esta implementación cada componente puede emitir unas señales propias acompañadas de datos pero semánticamente vacías: se emiten al darse una condición determinada en el componente pero tal condición o el estado del componente es irrelevante para la señal. Esta señal puede desencadenar una o más acciones en otros componentes, una vez que en el programa se establece esta relación denominada *conexión* entre señales y acciones. Este paradigma se puede ver en la programación de interfaces de usuario, donde el programador provoca «señales» al interactuar con botones y otros controles y los programas registran (establecen «conexiones») manejadores de eventos (métodos y funciones: las «acciones») para esas señales.

Esta construcción de señales y acciones (*signals and slots*) es una construcción de alto nivel que facilita la comunicación asíncrona entre componentes y la implementación del patrón *Observer* de programación (Gamma, Helm, Johnson, & Vlissides, 1995), por el cual los objetos observadores son notificados de un cambio de estado del objeto observado. Esta

construcción existe en el framework Qt[31] y lenguajes como C#, y también está disponible mediante bibliotecas en otros lenguajes.

4.1.5. Decisiones de compatibilidad y multiplataforma

Una de las mayores dificultades en la programación web cliente es conseguir la compatibilidad entre distintos navegadores. El desarrollo de los mismos sigue objetivos tanto tecnológicos como comerciales y la existencia de estándares reconocidos no garantiza que estos se cumplan, como tampoco que los usuarios dispongan de las últimas versiones de los navegadores. La computación móvil ha complicado aún más la foto, al requerir en muchos casos nuevas versiones reducidas de los navegadores.

En la implementación de APaM se ha tenido que priorizar el desarrollo para distintos navegadores con los siguientes objetivos en mente: que incorpore las funcionalidades deseadas, conseguir una aplicación web que soporte el máximo número de plataformas disponibles, y con una preferencia por el software libre frente al comercial o cerrado.

Con estos criterios, el mejor navegador es Firefox de la Fundación Mozilla: tiene versiones para los principales sistemas operativos de escritorio (Windows, Linux, OS X) y algunos de computación móvil (Android y Firefox OS, no habiendo actualmente versiones para iOS, Windows Mobile y otros); tiene un soporte muy completo de HTML5 y JavaScript; y es software libre. El navegador Chrome de Google es el segundo en prioridad, por su peor soporte de JavaScript (algunas características del estándar ECMA6 usadas en el desarrollo de APaM no están disponibles o requieren habilitar configuraciones expertas en Chrome) y no es software libre (aunque existe una versión libre, Chromium). El tercero en prioridad sería el navegador Safari de Apple, lo que aportaría compatibilidad con los dispositivos móviles basados en iOS (si bien existe una versión de Chrome para iOS, usa el mismo *engine* que Safari).

Estas diferencias entre navegadores afectan también a la biblioteca multimedia JsMIDI, que utiliza backends diferentes para Firefox, Chrome y otros navegadores, por lo que todo cambio en el backend requerirá un trabajo adicional para mantener la compatibilidad con otros navegadores.

31 Framework de desarrollo Qt, https://qt-project.org/

4.1.6. Decisiones de usabilidad y accesibilidad

Todo proyecto software debería considerar en algún momento cuán accesible a personas con capacidades diferente puede ser, especialmente si se pretende que tal software pueda ser usado por el mayor número de personas posible. La discusión no es baladí, pues se estima que hasta un 15 % de la población mundial padece algún tipo de discapacidad.[32]

Este proyecto ya parte de una selección previa del objetivo, que es hacia la *inteligencia musical* (de la teoría de las inteligencias múltiples de Gardner) de los niños y al aprendizaje *auditivo* (del modelo VAK de Dunn y Dunn), y de ahí las metáforas auditivas y la incorporación de elementos del mundo de la música.

Por otro lado, ya se ha planteado como objetivo la programación *visual* como elemento vehicular del aprendizaje frente a la programación mediante código fuente, que puede ser más fácilmente descrita por un lector de pantalla. Conviene comentar que existe una comunidad de desarrolladores con ceguera completa perfectamente capaces de desarrollar su trabajo con los entornos de desarrollo adecuados (normalmente complementados con lectores de pantalla e IDE que funcionan bien con los mismos).

Por todo lo anterior se van a tomar unos ciertos compromisos, que se pueden resumir en no introducir barreras arbitrarias y tratar de que internamente el HTML generado esté lo suficientemente anotado como para facilitar una incorporación de elementos de las etiquetas y elementos ayuda en un trabajo futuro. En detalle:

- Ya se ha decidido que la aplicación sea totalmente web y realizada con tecnologías HTML5. El seguimiento del estándar garantiza que los «cimientos» del proyecto sean accesibles. Aunque es cierto que aún puede haber una base instalada de productos de apoyo para web no compatibles aún con HTML5 muy grande, la tendencia es a que HTML5 se extienda a la mayor parte de la web nueva y ya existente y los productos de apoyo la incorporen también.
- Se ha comentado también que se desea evitar tecnologías «extrañas» como Flash, manteniendo todo en una misma tecnología.
- Los elementos que requieren de capacidad auditiva para la comprensión se han limitado a aquellos específicos del dominio musical.

32 Según datos de 2013 de la Organización Mundial de la Salud (http://www.who.int/mediacentre/factsheets/fs352/es/, consultado en junio de 2014).

- Todos los eventos que generen audio tienen también una representación visual distintiva. Por ejemplo, cada vez que se pulsa una tecla en el piano integrado, cambiará la representación de dicha tecla; cada vez que se reproduce una nota musical, el elemento que la reproduce cambiará de color.
- El color se usa como elemento estético y no como elemento semántico.
- Todos los elementos visuales (bloques y enlaces) están anotados en atributos accesibles vía DOM aunque tales anotaciones no sean visibles.

4.1.7. Adaptación del aprendizaje

Como ya se ha ido comentando, se ha tratado de centrar APaM en un dominio musical, tanto para aproximarse a la inteligencia musical de Gardner como a la parte auditiva del modelo VAK. Los motivos para esto han sido explicados en la parte correspondiente del estado del arte, y se resumen en que las herramientas de aprendizaje comparables a APaM en objetivos no consideran estos factores y se enfocan en muchos casos al movimiento de autómatas físicos o virtuales, a la creación de «historias» o a la programación de algoritmos de diversas formas.

La adaptación, en lo pedagógico y musical, se presenta de varias maneras:

- La incorporación de metáforas relacionadas con el mundo de la música.
- Bloques o elementos constructivos de las aplicaciones relacionados con este dominio: reproducción de melodías, tratamiento de notas musicales como datos, entrada de datos del usuario relacionada con el dominio (piano virtual, etc.), metrónomo, etc.
- Ejemplos de programas, ya implementados o propuestos al estudiante, dentro del mismo dominio: un «¡Hola, Mundo!» musical, adivinar la nota musical tocada, bucles para tocar escalas musicales, dirigir la reproducción musical interactuando con los elementos, etc.
- Otros detalles visuales de la interfaz relacionados con la música:
    - Iconos de la interfaz relacionados con el dominio donde es posible.
    - Los mensajes entre objetos se representan mediante notas musicales.

    - Los colores asociados a cada nota musical son los definidos en la escala Jameson, muy extendida para arte músico-visual (Collopy, Fuhrer, & Jameson, 1999; Jameson, 1844).

- Se incorporaron elementos que reflejan distintos aspectos del modelo Kolb alternando entre experimentación activa y observación reflexiva, y entre conceptualización abstracta y experiencias concretas:
    - En la narración aparecen preguntas que inciten a la reflexión.
    - Se intercalan ejemplos de programas con la narración para presentar los conceptos antes o después del ejemplo.
    - Se incorporan y sugieren actividades que obliguen al estudiante a ponerse manos a la obra y construir o terminar un programa.
    - Se incorporan y sugieren actividades en las que el estudiante tenga que analizar un programa y descubrir lo que hace.
- Se usan metáforas en la explicación, sugiriendo varias simultáneamente para cada concepto y explicando las diferencias con el concepto. Con esto se espera ayudar a los estudiantes a fijar mejor los conceptos, incitando a la reflexión y estimulándolos.
- La narración usa un lenguaje orientado a niños. Se usan caricaturas para representar tanto al narrador como al alumno.
- Los elementos constructivos de los programas también son caricaturas animadas, se denominan «criaturas» o Musicódigos (ilustración 4.1) y se relacionan también con el dominio.
- El tutorial presenta los contenidos de forma lineal reduciendo la capacidad de control de los alumnos, pues se espera que tengan pocos o ningún conocimiento previo sobre la programación (Pescador & González, 2013).

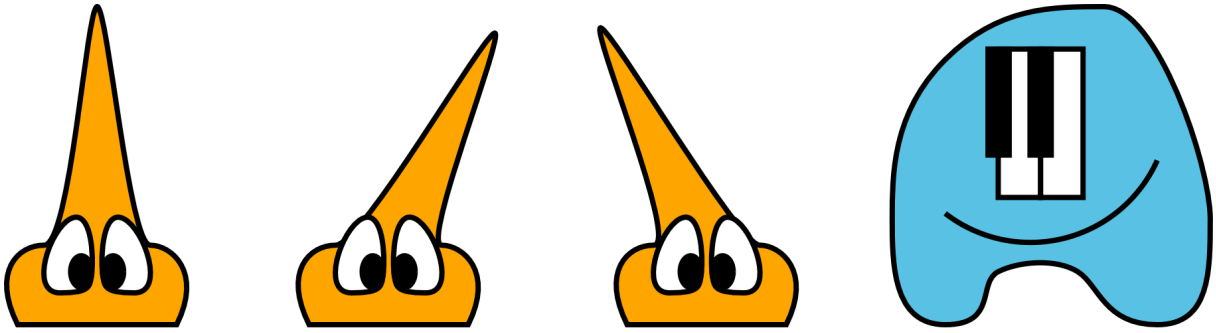


Ilustración 4.1: Ejemplos de criaturas «Metronomon» y «Pam»

4.1.8. Elementos de motivación al estudiante

A lo largo de la narración se han introducido elementos para mantener la motivación del niño. Estos elementos serán:

- Presentación progresiva de funciones. Los elementos constructivos de los programas, aunque inicialmente están disponibles, se van presentando poco a poco en el tutorial.
- Títulos por progreso. Al ir avanzando por el contenido guiado se va dando al estudiante un título honorífico que denote su progreso. Estos títulos pertenecen también al dominio: aprendiz, aficionado, músico, compositor, director de orquesta.

4.2. Diagrama de clases

En esta sección se incluyen varios diagramas de clases que reflejan cómo está programado APaM internamente.

A grandes rasgos, el proyecto se divide en las siguientes grandes partes:

- Los desarrollos propios en JavaScript, englobados en el objeto-espacio de nombres `ApamDesktop`.
- Bibliotecas JavaScript importadas. Cada una tiene una estructura propia que no se detalla.
- Los árboles DOM de los documentos principales.

Nótese que al ser JavaScript un lenguaje no tipado se han usado patrones de programación bastante flexibles que no dan lugar a una estructura de clases tan rígida y definida como ocurriría al programar en otros lenguajes como Java.

4.2.1. Estructura general de APaM

APaM se divide en los siguientes grandes bloques (ilustración 4.2):

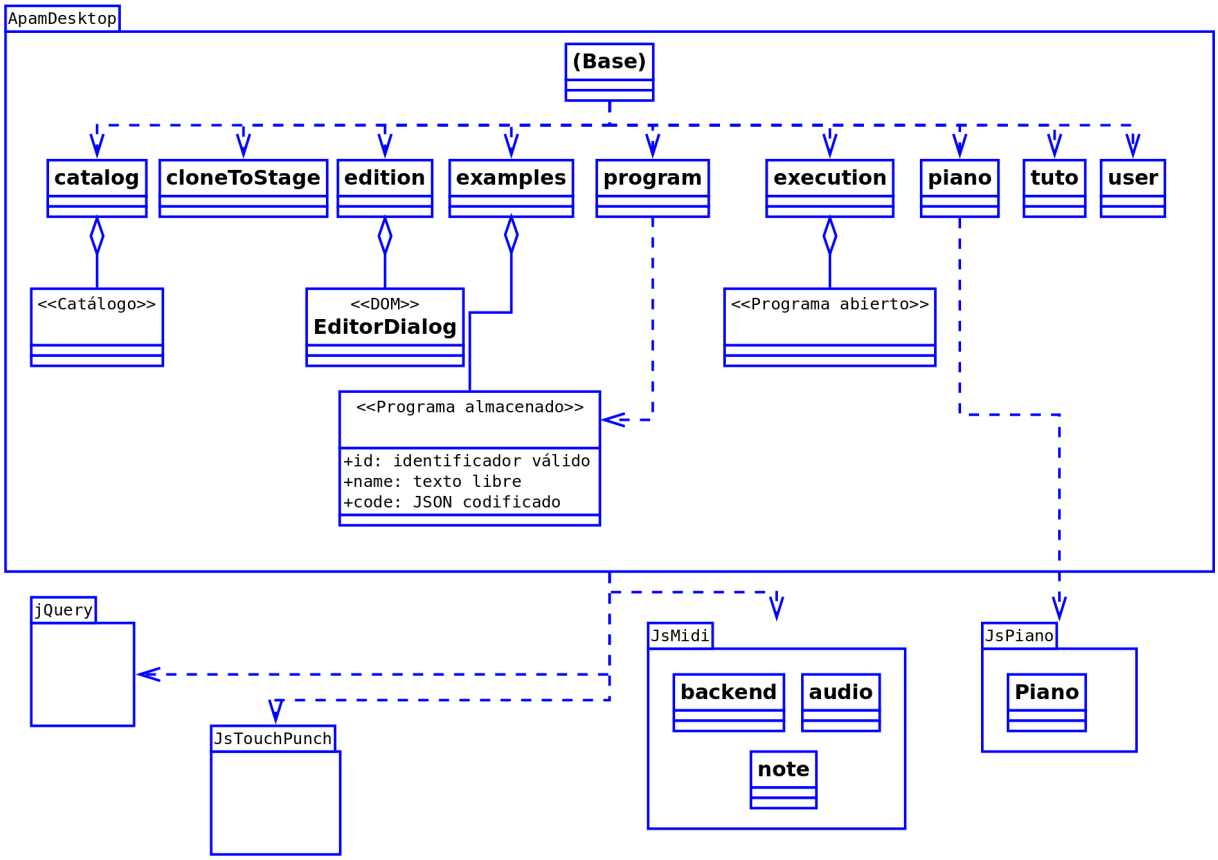


Ilustración 4.2: Estructura de APaM

- `execution`. Control de flujo durante la ejecución del programa, manejador de eventos y gestor de datos en tiempo de ejecución.
- `storeStage`. Mantenimiento del programa almacenado en memoria en un estado coherente con la interfaz de usuario, y almacenamiento y recuperación del programa en almacenaje permanente.
- `edition`. Manejadores de los editores de los tipos de datos reconocidos en APaM que están disponibles al usuario.
- `examples`. Gestión de los ejemplos precargados en APaM.
- `tuto`. Gestión del tutorial y automatización del mismo.

- `user`. Gestión de los datos de usuarios persistentes y de sesión. En el prototipo solo se guarda el título alcanzado, nombre y avatar del usuario en el tutorial.
- `piano`. Interacción con la biblioteca JsPiano.

4.2.2. Definición de los elementos constructivos

Los bloques o elementos constructivos de la programación visual, o criaturas como se denominan a lo largo del tutorial y narración, están definidos principalmente por tres categorías de características: señales, acciones o *slots* y propiedades o datos (ilustración 4.3).

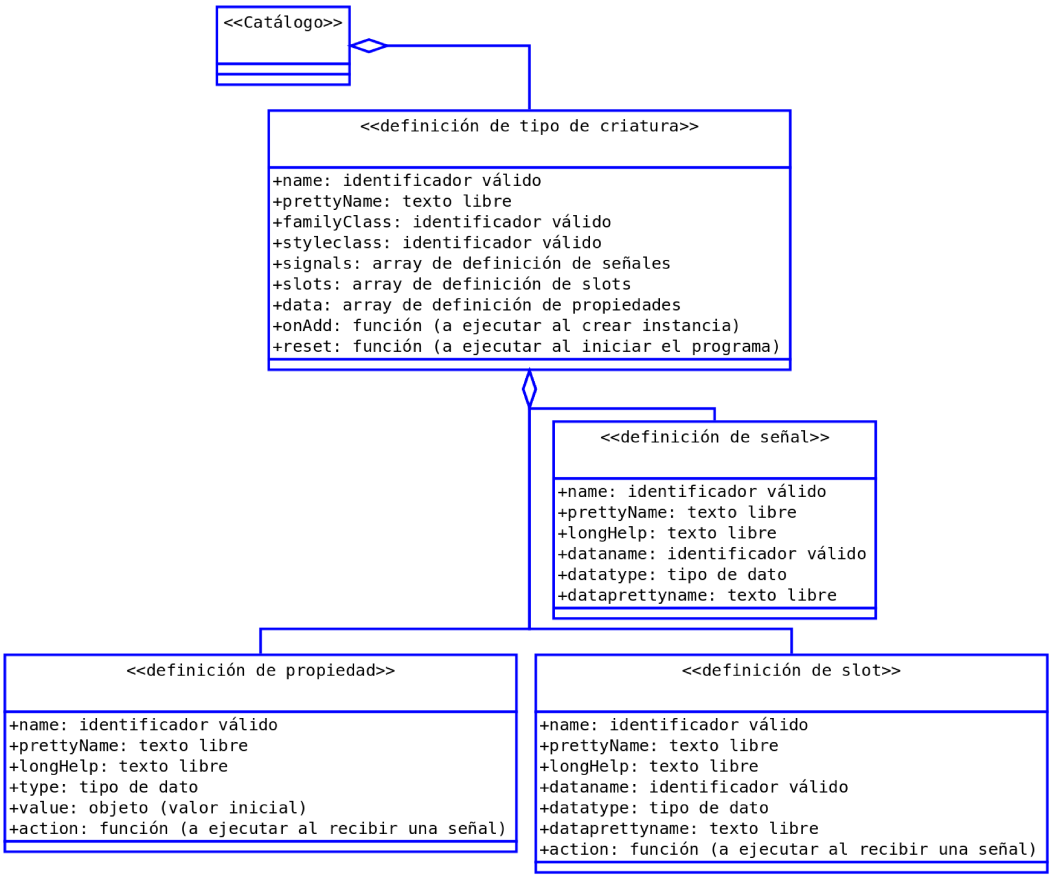


Ilustración 4.3: Definición de los elementos constructivos

4.2.3. Estructura de los programas

Los programas se definen de forma declarativa, estableciendo de qué criaturas se compone el programa, qué datos tienen y cómo se conectan entre sí a través de señales (ilustración 4.4). Nótese que la programación *declarativa* es diferente de la programación

imperativa en la que los programas se definen indicando qué acción se debe ejecutar en cada momento.

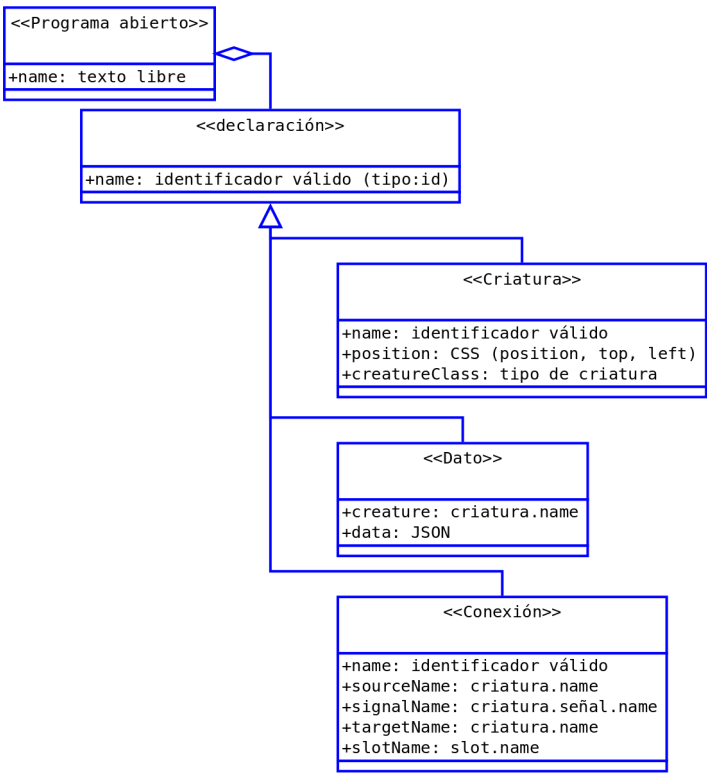


Ilustración 4.4: Estructura de los programas

4.2.4. Estructura DOM del documento HTML

La página principal de APaM, `apamdesktop.html`, se compone de las siguientes partes (ilustración 4.5):

- Una barra de opciones superior con el acceso a las «metafunciones» de programación, como cargar y guardar ficheros y el acceso a la ayuda.
- Una barra lateral a la izquierda, que muestra todos los elementos constructivos, es decir, el catálogo de criaturas, o mejor dicho, de *clases* de criaturas.
- Un área central donde se realiza la programación. A este área se arrastran mediante *drag & drop* criaturas desde catálogo, *instanciándose* en objetos particulares. Mediante menús que se abren en ventanas emergentes dentro del mismo documento se accede a las señales, acciones y datos de cada criatura. Mediante operaciones de *drag & drop*, arrastrando una señal desde el menú de señales de una

criatura hacia otra criatura y señalando la acción deseada, se pueden establecer conexiones entre las criaturas.

- Una barra lateral con los contenidos del tutorial y la ayuda interactiva (ver cursor gráfico azul y etiquetas verdes de avance).

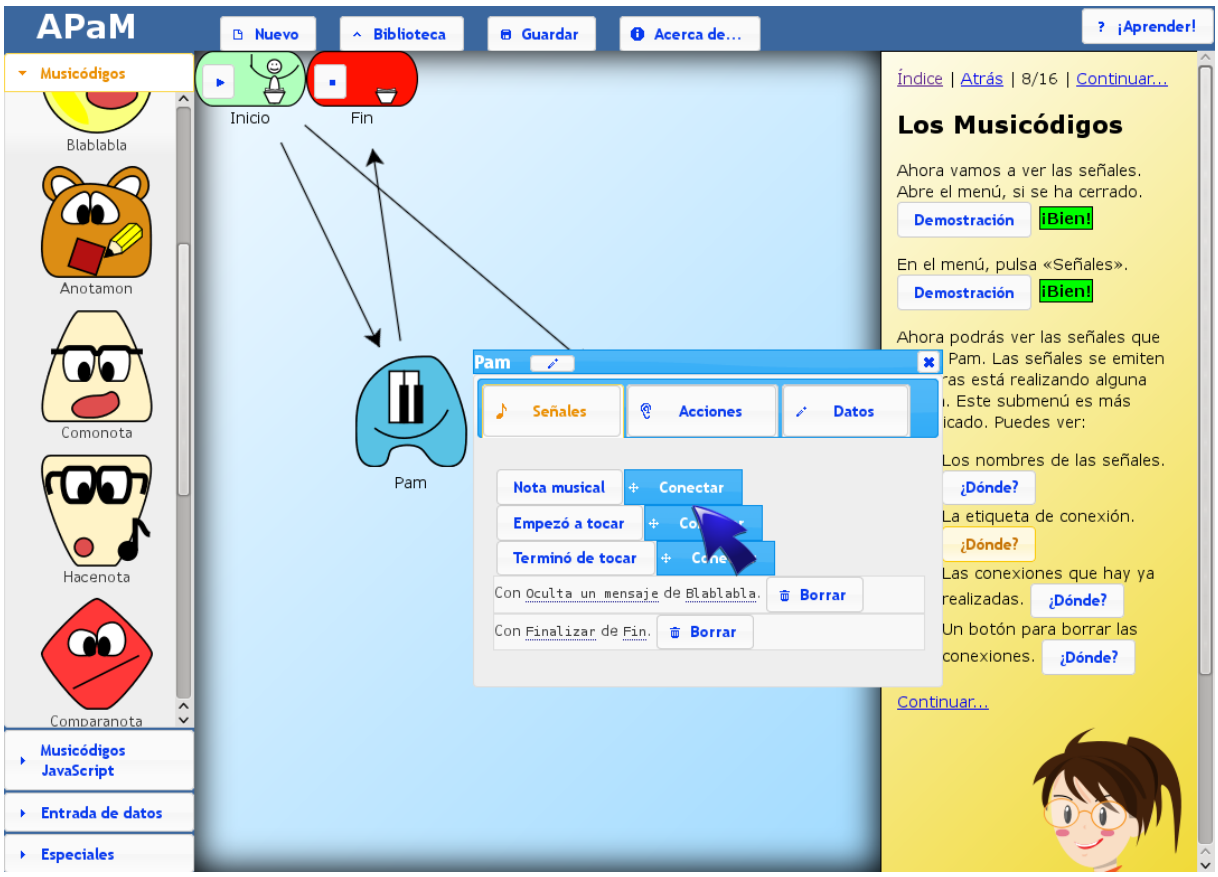


Ilustración 4.5: Pantalla de APaM mostrando el tutorial

5. Implementación y pruebas

En este apartado y los subsiguientes se describen algunos detalles de la implementación y del proceso de pruebas.

5.1. Análisis de alternativas tecnológicas

Para afrontar la fase de implementación con mayores garantías se utilizaron demos e implementaron algunos prototipos para validar el funcionamiento y compatibilidad de las funciones más relevantes.

5.1.1. Evaluación de JsMidi

Así, por ejemplo, con la evaluación de MIDI se pudo comprobar que la biblioteca JsMidi[33] funcionaba aceptablemente con los navegadores Firefox[34] y Chrome[35], tanto en escritorio como en Android[36], sin recurrir a plugins Flash o Java, es decir, puramente en HTML5. Sin embargo, la biblioteca tiene ciertas limitaciones:

- JsMidi trabaja de forma diferente en cada navegador, creando un *backend* distinto para cada plataforma. Así, Firefox y Chrome tienen distintos *backends*, y en navegadores sin soporte de HTML5 puede recurrir a Flash o Java. Es por ello que el funcionamiento de APaM puede variar dependiendo del navegador. En cualquier caso, como ya se ha discutido anteriormente, parece razonable utilizar HTML5 como *backend* y relegar la compatibilidad con navegadores antiguos o en entornos muy limitados a un trabajo futuro.
- JsMidi no utiliza las funciones MIDI del hardware o software si estuvieran disponibles, lo que permite su uso independientemente del dispositivo. Cada nota musical se almacena como una única *forma de onda* en un elemento `audio` de HTML5 y se reproduce a petición. Esto quiere decir que, a diferencia de en un sintetizador MIDI ideal, cada nota suena siempre igual independientemente de cuánto tiempo esté *pulsada* la tecla. Esta diferencia, que no es especialmente notable en un piano, es más clara en un instrumento de viento. En APaM no se han implementado medidas correctoras sobre este asunto, salvo no incluir otros instrumentos.

33 Biblioteca JsMidi de Michael Deal con licencia MIT, http://mudcu.be/midi-js
34 Navegador Mozilla Firefox, http://www.mozilla.org/firefox
35 Navegador Google Chrome, http://www.google.com/chrome/browser
36 Sistema operativo Google Android, http://www.android.com

- Además, como en JsMidi cada nota debe mantenerse en la memoria de trabajo del navegador, hay un límite práctico, especialmente en navegadores de móviles y tabletas, al número de instrumentos que pueden estar disponibles simultáneamente. En APaM solamente esta disponible un piano.
- Por último, JsMidi no genera las *callbacks* o notificaciones al programa que usa la biblioteca en los eventos de nota generada o reproducción terminada, requeridas para algunas funciones de APaM. La biblioteca ha sido modificada para incorporar estas notificaciones, para lo cual se han modificado los *backends* de HTML5.

5.1.2. Evaluación de JsPiano

Un enfoque distinto al de JsMidi es el de la biblioteca JsPiano de Peter Coles, con licencia MIT, que genera sonidos mediante funciones matemáticas sencillas (onda senoidal, cuadrada, triangular con decaimiento lineal, exponencial o abrupto). Esta biblioteca tampoco está exenta de limitaciones:

- Compatibilidad con otros dispositivos y navegadores menor que la de JsMidi.
- El audio suena significativamente más pobre e insulso. Los tonos tan puros tienden a resonar.
- Tampoco implementa por defecto un mecanismo para producir notas de diferente duración ni *callbacks*.

Por estas limitaciones se decidió prescindir de la generación de audio con esta biblioteca. Sin embargo, el aspecto visual e interacción con el piano es bastante satisfactorio y se reutiliza esa parte visual con pequeñas modificaciones (incluir los nombres de las notas musicales, sobeimpresas en cada tecla).

5.2. Descripción del software

En este apartado se describe el software creado y su funcionamiento, sin llegar a ser este un manual de usuario o una referencia.

5.2.1. Funcionamiento de APaM

APaM es una aplicación web que se ejecuta íntegramente en el navegador del equipo local. Desde el punto de vista del usuario, es un entorno de desarrollo integrado para programación visual, utilizando componentes visuales (las criaturas o Musicódigos) que se

comunican entre sí, reflejando las conexiones entre los componentes visuales el flujo de programa y de datos.

El usuario instancia los Musicódigos arrastrándolos desde la barra de Musicódigos hasta el área de programa. Cada tipo de Musicódigo posee unas «acciones» determinadas, como reproducir una melodía o comparar dos datos. Cada acción puede tener unos efectos perceptibles (visual o audiblemente) y además genera unas determinadas «señales», como por ejemplo, ha terminado de tocar la melodía o los dos datos son iguales. Además, los Musicódigos pueden almacenar datos de distintos tipos, como descripción de notas musicales o mensajes.

Los Musicódigos se comunican entre sí una vez que se establecen conexiones entre ellos. El usuario es quien mientras diseña el programa establece las conexiones entre las señales que emite un Musicódigo y las acciones de otro. Así pues, el Musicódigo «Inicio» tiene una señal «se ha iniciado el programa» que se puede conectar con «Pam» para que comience a reproducir una melodía; y se puede conectar la señal que emite este último al terminar la melodía con la acción de terminar la ejecución del programa del Musicódigo «Fin». Con esto se tiene un programa completo (ilustración 5.1) con un inicio, una operación y un final, en el que se resalta la estructura de programa dado que solo se utiliza la señal de reproducción terminada. En otros programas más complejos donde los componentes intercambien más datos entre sí el flujo de los datos cobra más protagonismo. Nótese que los componentes de inicio y fin tienen forma similar a los usados con esa misma función en diagramas de flujo tradicionales, y esto ocurre también con otros Musicódigos (tabla 5.1).

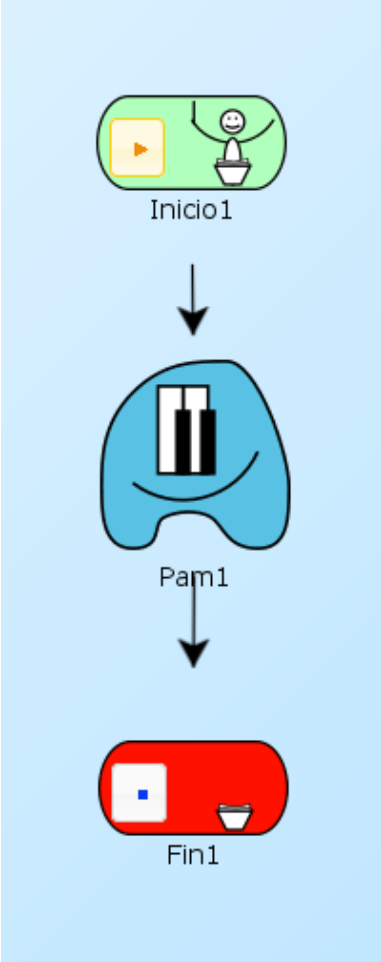
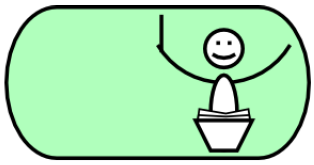
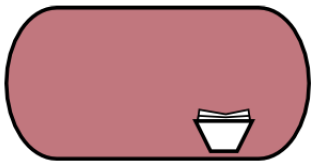
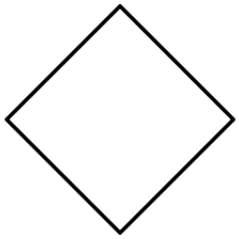
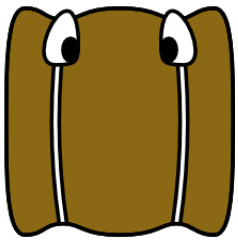
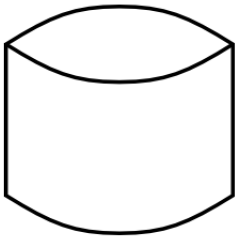


Ilustración 5.1: Ejemplo de programa en el que se resalta el flujo del programa

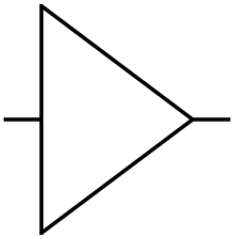
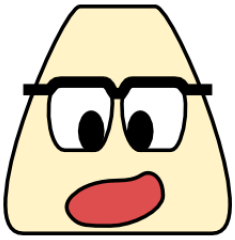
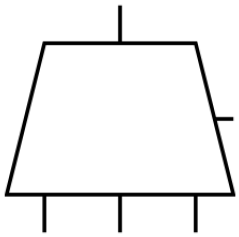
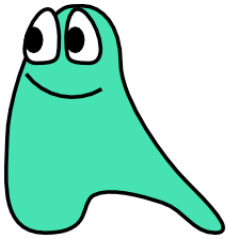
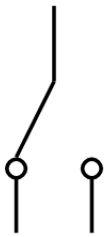
APaM cuenta con una biblioteca de programas propios a la que se van añadiendo los creados por el usuario. Los programas se pueden enviar por correo

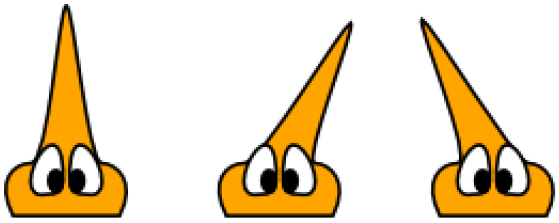
electrónico en forma de texto, un fichero con formato JSON[37] que contiene las descripciones de los Musicódigos usados y las conexiones entre ellos (ver texto 1).

Tabla 5.1: Analogías entre Musicódigos y elementos de otros dominios

Musicódigo	Diagrama de flujo
 Inicio  Fin	 Interrupción de terminal
 Compara	 Decisión
 Rutina	 Proceso predefinido
 Datos	 Disco

37 Notación JSON (JavaScript Object Notation), http://json.org/

Musicódigo	Diagramas de electrónica y señal
 Transforma	 Ganancia
 Comonota (desglosa una nota)	 Demultiplexor
 Cambion	 Interruptor biestable

Musicódigo	Objetos del mundo real
 Metronomon	 Metrónomo [1]
 Piano [2]	 Piano [3]

1 Imagen de Paco de Badajoz, España. Licencia Creative Commons Attribution 2.0.

2 Imagen de dominio público de OpenClipart.org.

3 Imagen © Copyright Steinway & Sons. Licencia Creative Commons BY-SA 3.0.

```
[ {"method":"addCreature","name":"Inicio1","creatureClass":"inicio",
   "position":{"position":"absolute","left":"499px","top":"154px"}} ,
  {"method":"addCreature","name":"Fin1","creatureClass":"fin",
   "position":{"position":"absolute","left":"500px","top":"493px"}} ,
  {"method":"addCreature","name":"Pam1","creatureClass":"bom",
   "position":{"position":"absolute","left":"501px","top":"293px"}} ,
  {"method":"addConnection","sourceName":"Inicio1","signalName":"start",
   "targetName":"Pam1","slotName":"start",
   "connectionName":"addConnection:connection0"} ,
  {"method":"addConnection","sourceName":"Pam1","signalName":"finished",
   "targetName":"Fin1","slotName":"end",
   "connectionName":"addConnection:connection1"} ]
```

Texto 1: Código del programa de ejemplo

5.3. Tutorial

El aprendizaje de los conceptos de programación y del uso de la herramienta está guiado por un tutorial interactivo, que se dirige al usuario, presumiblemente un niño, de forma directa, intercalando preguntas al usuario y respondiendo a posibles preguntas del mismo. El tutorial guía al usuario paso a paso en la creación de los primeros programas, comprobando que el usuario está haciendo las operaciones pedidas por el tutorial mediante etiquetas que se encienden cuando se cumplen las condiciones esperadas, y demostrando con animaciones cómo se llevan a cabo con el ratón.

5.4. Otros contenidos

El tutorial se puede consultar fuera del programa APaM e imprimir en papel, aunque lógicamente se pierden los formatos e interactividad. Técnicamente, es APaM el que paresa los fragmentos y scripts incluidos en el fichero de tutorial y los presenta en APaM.

También se ha incluido una referencia de los Musicódigos con la descripción de sus acciones, señales y datos, generada de forma a partir de las descripciones de los mismos.

Por último, también se ha creado una página de entrada[38] con la información de portada del proyecto (título, resumen, nombre del grado y universidad, autor y director, etc.) e información de contacto.

38 Página de entrada a APaM, http://apam.esy.es

5.5. Metodología de pruebas

A la hora de realizar un desarrollo software uno de los mecanismos imprescindibles para asegurar la calidad del producto son las pruebas que se realizan a lo largo del mismo, garantizando que todo funciona de forma repetitiva y que los desarrollos nuevos no interfieren en los desarrollos anteriores (lo que se conoce como errores de regresión). Es conveniente establecer una política de pruebas antes de empezar el desarrollo del mismo.

En proyectos de cierto tamaño, se suele establecer un mecanismo de pruebas automatizadas (por ejemplo JUnit), que funciona a grandes rasgos haciendo pruebas de «caja negra», enviando un juego de datos conocido como entrada a cada parte del código del programa y comparando las salidas del mismo contra otro juego de datos de salida esperados. Pero el coste de preparación de dichos juegos de datos debe balancearse contra el tiempo de hacer las pruebas «a mano», introduciendo datos manualmente e inspeccionando los resultados.

En un proyecto como este hay unas características que dificultan la aplicación de este enfoque automatizado:

- Hay mucha incertidumbre en lo que es posible hacer y lo que no. Desarrollar los juegos de datos *a priori*, que es lo correcto desde el punto de vista metodológico, no es efectivo en cuanto a coste.
- No hay un equipo de programadores donde se pueda obtener un beneficio de un reparto de tareas, entre diseñadores y mantenedores de pruebas y desarrolladores.
- El factor tiempo juega un papel importante y los requisitos pueden cambiar tras una evaluación de usabilidad o tras realizar un prototipo. Los juegos de datos podrían cambiar con frecuencia.
- El proyecto tiene un foco muy importante en la interacción con el usuario, más que en los algoritmos en sí, y preparar juegos de datos adecuados para interfaces de usuario, incluso en HTML5 donde se dispone de un árbol DOM que se puede verificar programáticamente, tiene un coste significativo.
- Las bibliotecas para hacer pruebas automatizadas no están tan estandarizadas como en otros lenguajes. Hay algunas disponibles como JsUnit o JsTester. La Fundación Mozilla entre las ayudas a desarrolladores sugiere un catálogo de herramientas para automatizar tests para cada situación particular), pero las que se han localizado

parecen tener un ámbito de aplicación muy específico o llevan años sin mantenimiento. El coste de implantarlas para un número pequeño de casos puede no merecer la pena.

Por todo lo anterior, las pruebas a lo largo del desarrollo se han realizado de forma manual siguiendo un guión que incluye las siguientes tareas.

Primero, se realiza un diseño de un número de ejemplos de programa en los que se usa la mayor parte de funcionalidad disponible en APaM. Algunos de estos ejemplos se han hecho disponibles en la biblioteca de APaM.

Los ejemplos se crean y almacenan en la biblioteca de APaM. En cada evento de pruebas, se realiza la carga y ejecución manual de los ejemplos, y una comprobación visual de que los resultados obtenidos son los esperados: que se emiten todas las señales, que todos los Musicódigos funcionan correctamente, que los Musicódigos animados cambian de estado, que los datos se almacenan y transfieren de forma correcta entre Musicódigos, que el programa llega a la condición de finalización normalmente, etc.

También se realiza en cada evento de pruebas la implementación de los ejemplos. Aunque el programa ya esté creado, se recrea nuevamente desde cero, comprobando que todas las conexiones se pueden establecer con normalidad, que se pueden editar y dar valor a los datos, y que al ejecutar el programa los resultados son idénticos al del programa almacenado.

La última parte consiste en la ejecución manual de los tutoriales. Los tutoriales contienen elementos interactivos, como demostraciones de la interfaz de usuario en la que un cursor animado realiza las operaciones de arrastre y pulsaciones o señala elementos; comprobaciones de que el usuario avanza y realiza las tareas adecuadamente; carga de soluciones de programas; asignación de títulos; y preguntas al usuario y cuestionarios autocorregidos. En las pruebas manuales se comprueba que todos funcionan correctamente.

Los eventos de pruebas se han realizado en varios hitos a lo largo del desarrollo. En los hitos menores, como añadido de alguna funcionalidad adicional concreta, se realiza una prueba parcial que contempla solo algunas tareas o ejemplos de las indicadas, incluyendo siempre alguna prueba específica de retrocompatibilidad.

6. Validación

En este apartado se han detallado los resultados del estudio y validación de usabilidad y aplicabilidad al aprendizaje del software desarrollado.

Para realizar este estudio se han implementado dos medidas:

- Diseño de un cuestionario de usabilidad (ver Anexo I) y aplicabilidad al aprendizaje que contiene preguntas para valorar cada ítem y un cuadro de observaciones.
- Observación directa del uso por usuarios niños por parte de los expertos.

El cuestionario contiene un número de afirmaciones en distintas categorías a valorar el grado en que se cumplen en una escala sumativa. Habrá además un campo de texto libre para formular comentarios. El cuestionario contiene una compilación de preguntas genéricas (Donoso, 2010) aplicables a casi cualquier software educativo con uso en el aula, tratando así de conseguir una evaluación más completa y sin sesgo.

Este cuestionario se ha entregado a validadores expertos en distintos perfiles (en educación y pedagogía, en programación y tecnología, etc.). Esta validación ha permitido detectar carencias en distintas áreas y obtener sugerencias sobre el trabajo futuro. Por otro lado, la observación directa ha permitido observar en qué puntos tienen más dificultad los niños y cómo mejorar la aplicación.

Se consideró el usar otros mecanismos de evaluación de usabilidad como herramientas de tracking, de las que hay muchos tipos: seguimiento ocular, que requiere hardware específico para detectar el movimiento del ojo; seguimiento de ratón, que detecta los movimientos y clics de ratón y los agrega en «mapas de calor», como ClickHeat, ClickTale, Clixpy, etc.; o grabadores de pantalla y sesión. Estos mecanismos de evaluación más avanzados han sido descartados por distintos motivos:

- La interactividad de APaM (no es una página web, sino un minientorno de desarrollo con cantidad de contenido dinámico) dificulta las aproximaciones convencionales de grabación de clics y tratamiento estadístico como mapas de calor.
- Al ser un software orientado a niños surgen complicaciones para hacer pruebas de validación concretas, basadas en la ejecución de tareas específicas, como suele hacerse con otros tipos de páginas con procesos más claros (registrarse en una web, comprar un libro, etc.).

- La grabación completa de sesiones mediante capturas de pantalla, aunque es viable siempre que se pueda tener acceso físico a los equipos (se complica al operar en remoto), requiere la inspección visual de gran cantidad de datos. Dadas las restricciones en recursos y alcance de este trabajo, se descarta este mecanismo.

6.1. Estudio de la aplicabilidad a la enseñanza

El cuestionario de evaluación contiene categorías específicas sobre la aplicabilidad del software a los alumnos objetivo, educadores y entorno educativo, teniendo en cuenta la existencia de diferentes estilos y metodologías tanto de aprendizaje como docentes, y afrontando la realidad educativa en distintos casos de uso (escolar, individual, etc.).

El cuestionario ha sido entregado de dos formas, una adecuada para impresión con el cuestionario en formato de documento de texto similar al que se puede ver en el anexo; y otra en forma digital para responder en el ordenador, a través de un formulario de Google Drive. Los enlaces a ambos formatos han sido publicados en varios grupos de Facebook y Twitter, y enviados a conocidos a través de correo electrónico u otra mensajería.

En total se han obtenido solamente seis validaciones en cuestionario[39], más algunos comentarios de forma verbal o escrita sin cuestionario. Algunos factores que pueden haber influido en el número bajo de validaciones son las fechas en las que se ha llevado a cabo, en julio y con poco margen de tiempo; los requisitos de APaM, que requiere unos navegadores determinados que pueden no estar instalados en los equipos de los usuarios, y que tampoco puede ser evaluado fácilmente en un móvil o tableta pequeña; y la publicación de APaM y su cuestionario en un número pequeño de grupos donde la herramienta sea de mayor interés.

Con respecto al alumno (preguntas del bloque 1 del cuestionario) las valoraciones recibidas son bastante positivas, con promedios de 4 o superior, en una escala de 1-peor a 5-mejor. Hay prácticamente unanimidad en las respuestas a la pregunta 1.5: el uso del software resulta motivador al alumno, y también promedios muy próximos a 5 en la pregunta 1.1, ajuste al perfil del alumno. Los validadores han reportado usar el software con niños de 9 años, comentando que con el soporte adecuado del profesor podría ser usado por niños de hasta 5 años. Otro experto destacó que el estilo y apariencia del software resultó también motivadora para una alumna de 17 años.

39 Resultados de la validación, https://drive.google.com/file/d/0B9GATt8-esradXFLVDJGNnktQWM

En cuanto a las capacidades de uso del software por parte del profesor (bloque 2) los resultados son más bajos. Por ejemplo, en cuanto al uso del software de forma colaborativa (pregunta 2.7) y la integración del software con otras herramientas software y hardware (2.8 y 2.9), los promedios obtenidos están entre 3 y 3,5. Otros resultados mejorables se obtienen en las preguntas 2.3 (el software se adapta a la experiencia del profesor) y 2.6 (profesor capacitado para integrar el software en el programa docente), con resultados entre 3,5 y 4. Se destacan los resultados obtenidos en la pregunta 2.5, en la que el profesor se declara dispuesto a recibir la capacitación necesaria para usar el software, con un promedio de 4,5.

En la categoría del entorno (bloque 3) se obtienen las puntuaciones más bajas, indicando falta de equipos o equipos inadecuados y entorno mejorable: promedios entre el 3 y el 3,5.

El bloque 4.1 se centra en evaluar el software en cuanto a sus contenidos y funciones, con promedios altos (por encima de 4) en la mayoría de preguntas, destacando la metodología del programa (4.1.2), que facilita el aprendizaje (4.1.5) y es interactivo (4.1.6). Se obtiene valoración más baja en la pregunta relativa a la evaluación del alumno (4.1.3), esperable quizá ya que no hay un mecanismo de evaluación definido, salvo las preguntas de autoevaluación incluidas en el tutorial, cuyos resultados no se almacenan o exportan, y la consecución del título más alto que puede lograrse simplemente avanzando todas las páginas del tutorial. Este será otro punto de mejora.

En cuanto a la guía de uso del software (bloque 4.2) se obtienen promedios entre 4 y 4,5. Aquí se obtienen resultados quizá por encima de lo esperado, si se considera que no hay un «kit educativo» que acompañe al software más allá del propio tutorial. Sin duda, uno de los puntos de mejora más importantes será el desarrollo de dicho kit, respondiendo también a las carencias detectadas en el bloque 2 acerca de las capacidades iniciales de los profesores para usar el software con efectividad.

En el bloque 4.3 se trata la adaptabilidad del software, con buenos promedios por encima de 4, destacándose que el software permite la experimentación libre (pregunta 4.3.2) y que permite expresar la creatividad del alumno (4.3.4).

Los expertos aportaron también otros comentarios de interés, destacando la necesidad de materiales TIC integrables en el currículo y con una metodología constructivista, y la importancia del trabajo colaborativo, un punto mejorable de APaM.

6.2. Estudio de usabilidad y accesibilidad

Los objetivos de usabilidad se valoran en cuanto a la facilidad de uso de la aplicación en general y la capacidad para usarlo en los entornos educativos.

La batería de preguntas sobre usabilidad (7 preguntas distintas) fue respondida con promedios por encima del 3,5. La puntuación más baja se refiere al uso por parte de personas con capacidades diferentes (pregunta 4.4.7), resultado esperable dado el enfoque del programa. La pregunta 4.4.6 sobre la respuesta a errores también puntuó en promedio por debajo de 4. Por último, la pregunta 4.4.1 acerca de la curva de aprendizaje inicial también puntuó por debajo de 4. Sin embargo, las respuestas a la pregunta sobre recordar la forma correcta de uso posteriormente tienen puntuaciones satisfactorias. Estos resultados sugieren una revisión de la interfaz de usuario, en particular de las operaciones de borrar y deshacer e incluyendo otras funciones de copiar y pegar, etc.

Adicionalmente, se realizó una revisión técnica utilizando un validador[40] de HTML y CSS, comprobando la validez del código creado. Hay que tener en cuenta dos factores a la hora de valorar estos resultados: que los validadores de HTML5 están poco maduros y se encuentran muchas veces en un estado «experimental»; y que al tratarse APaM de una aplicación interactiva con mucho contenido dinámico y generado por JavaScript, los resultados que se obtengan con validadores automáticos basados en *snapshots* (es decir, en el estado estático de la página tras su carga) serán poco indicativos de los problemas reales de usabilidad o accesibilidad de la aplicación.

Los resultados de la validación de HTML5 han sido positivos (*Passed, 1 Warning: Using experimental feature: HTML5 Conformance Checker*) para todas las páginas del proyecto: la página de entrada, la aplicación web APaM, el tutorial, el acerca-de y la referencia del catálogo de criaturas. Los resultados de la validación de CSS3 dan errores en las hojas de estilo utilizadas en la página de entrada (realizada con una herramienta propiedad del servicio de hosting; provoca más de 600 errores) y en las hojas de estilo de jQuery. Estos errores no se han intentado corregir ya que las hojas de estilo con errores han sido generadas por herramientas externas al proyecto y se ha comprobado que la visualización es correcta en distintos navegadores.

40 Validador de marcaje de HTML5 de W3C, http://validator.w3.org/

7. Conclusiones

En este capítulo se describen los objetivos cumplidos y los que no se pudieron completar, se valoran los resultados de la validación y se detallan las propuestas de trabajo futuro resultantes de todo lo anterior.

7.1. Objetivos alcanzados

Este trabajo describe, de acuerdo con los objetivos marcados en el apartado 3.1, las motivaciones para introducir la programación a niños, una selección de herramientas existentes con un fin similar y finalmente la elección de un enfoque educativo innovador con el que desarrollar una nueva herramienta, APaM: Aprende Programación Haciendo Música. También recogemos en el trabajo el proceso de análisis, diseño e implementación de la herramienta, describiendo su manejo y funciones. Por último, realizamos una validación desde los puntos de vista educativo y tecnológico de la herramienta recogiendo y comentando los resultados de la misma.

Los objetivos particulares de APaM se pueden englobar en tres categorías: funcionales, educativas y técnicas. Con respecto a los primeros, se ha conseguido crear la herramienta deseada de programación visual que puede ser usada por niños y con una ambientación musical. Planteada como un IDE en miniatura da gran libertad al usuario para desarrollar y expresar su creatividad. También se ha podido integrar el tutorial interactivo para guiar al usuario en los primeros pasos. Este tutorial, soportado por las capacidades del IDE, ha conseguido introducir estos conceptos de programación:

- Conceptos generales sobre la programación: qué es un programa y qué significa programar.
- Conceptos básicos sobre orientación a objetos.
- Conceptos de programación estructurada.
- Tipos de datos básicos y agregados.
- Programación visual, tratando el flujo de programa y de datos.
- Introducción a la programación mediante lenguajes de código fuente.

Desde el punto de vista técnico se ha conseguido que APaM aporte una funcionalidad básica íntegramente en cliente, y gracias a la utilización de tecnologías estándares HTML5 compatibles con Firefox y Chrome, los navegadores más extendidos, se

soporta la práctica totalidad de dispositivos de escritorio (Windows, Linux y Mac) y una amplia base dispositivos móviles (dispositivos con Android).

Por último, los resultados de la validación han dado resultados bastante satisfactorios, probando que:

- La herramienta cumple su función principal de facilitar el aprendizaje de los conceptos de programación por niños.
- La herramienta resulta motivadora, tanto para el estudiante como el profesor.
- La herramienta permite la experimentación libre y fomenta la creatividad.
- La herramienta satisface unos niveles de usabilidad adecuados al uso que se le quiere dar.

7.2. Discusión y trabajo futuro

Un proyecto de este tipo puede crecer y desarrollarse por muchos caminos en los mismos tres ámbitos mencionados antes.

7.2.1. Mejoras funcionales

No fue posible incorporar al desarrollo la funcionalidad necesaria para mostrar otros conceptos importantes de programación como la herencia en el paradigma de orientación a objetos, o la recursión.

En el apartado musical también hubo que aceptar ciertas limitaciones, como la falta de otros instrumentos musicales y de otros medios de interacción, como la *batuta*. Tampoco se han proporcionado métodos para controlar el volumen o el *tempo* de la reproducción. Otras funciones interesantes que se pueden añadir para enriquecer APaM es el trabajo con formatos de onda, importados del ordenador o grabados por el micrófono del dispositivo. Una muestra de las posibilidades las da el proyecto WebAudio[41]: visualización de onda, espectro en frecuencia y ecualización, detección de tono, vocoders, etc.

7.2.2. Mejoras técnicas

En la implementación de la herramienta se han dejado de lado muchos aspectos de usabilidad y accesibilidad. El dominio musical supone un conflicto con las personas con discapacidad auditiva. La programación visual y la interacción basada en ratón complica el uso a personas con discapacidades visuales.

41 Proyecto WebAudio de Chris Wilson con licencia MIT, http://webaudiodemos.appspot.com/

Otro aspecto importante es mejorar las características de multiplataforma e independencia del dispositivo. Aunque se tratará de elegir tecnologías tanto estándares como extendidas, la extrema diversidad de dispositivos y plataformas hace muy costoso hacer ciertos desarrollos de forma totalmente agnóstica. El siguiente paso debería ser conseguir la compatibilidad con los navegadores propios de iOS basados en WebKit, con lo que se conseguiría cubrir una base de dispositivos muy extendida que es la de tabletas y móviles de Apple.

7.2.3. Mejoras de ámbito educativo

Para conseguir un aprendizaje colaborativo, ya fuera en el aula o sobre comunidades online, sería necesario dotar a la herramienta de funciones de comunidad virtual de aprendizaje, trabajo en grupo, integración con redes sociales, etc. En el prototipo solo se ha incluido una función básica para exportar el código del programa y enviarlo como texto por email o cualquier otro canal. Aunque este tipo de funciones son sumamente interesantes desde el punto de vista educativo, en el prototipo se ha decidido concentrar los esfuerzos en el trabajo individual. Además, esto trae otros requisitos de mayor nivel, tanto en lo tecnológico (*hosting* de la aplicación y base de datos, *middleware* de intercomunicación, protocolos de comunicación en el cliente, etc.), como metodológico (cómo integrar la aportación de otros miembros de la comunidad) y éticos (protección de la privacidad del menor en un entorno difícil de controlar como Internet). Aunque todos estos asuntos tienen respuesta, se ha preferido dejar estas como trabajo futuro.

Teniendo APaM una finalidad educativa, parece inmediato considerar hacer la aplicación accesible como objeto SCORM (*Shareable Content Object Reference Model*). Esta integración puede hacerse a varios niveles. El nivel más bajo sería encapsular APaM en un objeto SCORM sin más, intercambiando la mínima información con la plataforma de aprendizaje. Los niveles más altos implicarían el intercambio de datos de usuario y de resultados de aprendizaje. Para realizar esta integración de alto nivel habría que definir los elementos que se van a intercambiar y el uso que se va a hacer de los mismos, cuáles son los resultados de aprendizaje más relevantes y cómo exportarlos de forma significativa a la plataforma de aprendizaje, además de compartimentar las lecciones del tutorial. En esta primera versión de APaM se ha descartado hacer este estudio. Sin embargo, en el estado actual APaM es muy fácil de, al menos, encapsular en un objeto educativo de SCORM (SCO), pues todos los recursos necesarios para el funcionamiento del IDE pueden desplegarse juntos en un paquete y no se requiere de procesamiento en servidor.

También es posible profundizar y mejorar el tutorial, realizando un diseño instruccional más exhaustivo, contemplando otras posibilidades de adaptación e interacción por el usuario, incluyendo otro tipo de recursos multimedia e interactivos, o elementos de motivación como logros y medallas.

Por último y no menos importante, la validación reveló importantes carencias tanto en el entorno de uso en las aulas como de la falta de capacitación de los profesores para integrar el software en el programa docente, en actividades colaborativas y de forma integrada con otros programas y herramientas. Por ello, un trabajo muy importante a realizar es acompañar APaM de un «kit para educadores» que permita la formación de estos en el uso del software en el aula, la integración del software en el entorno escolar y también recomendaciones de uso y actividades para uso colaborativo y fácil integración en el programa docente.

I. Cuestionario de usabilidad

Indique el grado de acuerdo con la afirmación expresada, donde «1» es en total desacuerdo y «5» es en total acuerdo. Si no aplica al software evaluado no marque ningún grado. Utilice el cuadro de observaciones al final para cualquier otro comentario.

Tabla I.1: Cuestionario de usabilidad y aplicabilidad

	Indicadores	**Grado**
1.	**Alumno**	
1.1.	El software se ajusta al perfil académico del alumno.	①②③④⑤
1.2.	El software se ajusta a la edad del alumno.	①②③④⑤
1.3.	El software se adapta al estilo de aprendizaje del alumno.	①②③④⑤
1.4.	El software está adaptado a la experiencia previa y conocimientos en el uso del ordenador por parte del alumno.	①②③④⑤
1.5.	El uso del software resulta motivador al alumno.	①②③④⑤
2.	**Profesor**	
2.1.	El software se ajusta al currículum del profesor.	①②③④⑤
2.2.	El software se adapta al estilo docente del profesor.	①②③④⑤
2.3.	El software está adaptado a la experiencia previa y conocimientos en el uso del ordenador por parte del profesor.	①②③④⑤
2.4.	El uso del software resulta motivador al profesor.	①②③④⑤
2.5.	El profesor está dispuesto a recibir la capacitación necesaria para usar de forma eficaz el software.	①②③④⑤
2.6.	El profesor está capacitado para integrar el software en el programa docente.	①②③④⑤
2.7.	El profesor está capacitado para trabajar con el software de forma colaborativa o grupal.	①②③④⑤
2.8.	El profesor está capacitado para usar el software de forma integrada con otras herramientas software disponibles en el entorno.	①②③④⑤
2.9.	El profesor está capacitado para usar el software de forma integrada con los equipos hardware disponibles en el entorno.	①②③④⑤
3.	**Entorno**	
3.1.	Los equipos informáticos donde se ejecutará el software son adecuados para su uso con normalidad.	①②③④⑤

	Indicadores	**Grado**
3.2.	El entorno donde se usará el software es ergonómico y permite trabajar sin distracciones.	①②③④⑤
3.3.	Existen suficientes equipos informáticos para soportar la actividad docente con el software.	①②③④⑤
4.	**Aplicabilidad del software**	
4.1.	**Contenido**	
4.1.1.	Los contenidos que imparte el software son adecuados para el alumno y programa docente.	①②③④⑤
4.1.2.	La metodología que usa el software es adecuada para el alumno y programa docente.	①②③④⑤
4.1.3.	El mecanismo de evaluación del alumno es adecuado.	①②③④⑤
4.1.4.	Los contenidos dan el *feedback* adecuado al alumno ante fallos y respuestas correctas o incorrectas.	①②③④⑤
4.1.5.	El uso del software facilita el aprendizaje.	①②③④⑤
4.1.6.	El software es interactivo y responde a la interacción del alumno.	①②③④⑤
4.2.	**Guías de uso**	
4.2.1.	El software proporciona o trae apoyo para orientar al profesor en su uso docente.	①②③④⑤
4.2.2.	El software proporciona o trae apoyo para facilitar al alumno el uso.	①②③④⑤
4.2.3.	La ayuda del software o material de apoyo es comprensible y útil.	①②③④⑤
4.3.	**Adaptabilidad**	
4.3.1.	El software y los contenidos se pueden usar bajo distintos estilos docentes.	①②③④⑤
4.3.2.	El software y los contenidos se pueden usar con alumnos con distintos estilos de aprendizaje.	①②③④⑤
4.3.3.	El software permite la experimentación libre por parte del alumno.	①②③④⑤
4.3.4.	El software permite expresar la creatividad del alumno.	①②③④⑤
4.4.	**Usabilidad**	
4.4.1.	El software es intuitivo y la curva de aprendizaje inicial es asequible.	①②③④⑤
4.4.2.	El software es intuitivo y la forma correcta de usarlo es fácil de recordar.	①②③④⑤
4.4.3.	La estructura del software es comprensible.	①②③④⑤

	Indicadores	**Grado**
4.4.4.	El software no distrae del aprendizaje de los conocimientos, habilidades o actitudes esperados.	①②③④⑤
4.4.5.	La información necesaria para operar el software es fácilmente accesible.	①②③④⑤
4.4.6.	Los errores cometidos usando el software se arreglan fácilmente.	①②③④⑤
4.4.7.	El software puede ser usado por personas con capacidades diferentes.	①②③④⑤

5. Observaciones

(Indique si aplica la pregunta a la que hace referencia. Describa brevemente características del alumno y profesor [edad, currículum, estilos, metodología usada, etc.].)

Bibliografía

Begel, A. (1996). *LogoBlocks: A graphical programming language for interacting with the world*. *Electrical Engineering and Computer Science Department, MIT, Boston, MA*. MIT, Boston, MA.

Brown, P. H. (2008). Some field experience with Alice. *Journal of Computing Sciences in Colleges*, *24*(2), 213–219.

Cheeseman, T., Nguyen, N., & Osecki, J. (2009). *Generic Web Services Mash-up Integrated Development Environment*. *gicl.cs.drexel.edu*.

Collopy, F., Fuhrer, R. M., & Jameson, D. (1999). Visual music in a visual programming language. In *Visual Languages, 1999. Proceedings. 1999 IEEE Symposium on* (pp. 111–118). doi:10.1109/VL.1999.795882

Cooper, B. B. (2012). Why Codecademy is overrated and missing its target audience. *Attendly Blog*. Recuperado el 20 de julio de 2014, de http://www.attendly.com/why-codecademy-is-overrated-and-missing-its-target-audience/

Cox, P. (2008). Visual programming languages. *Wiley Encyclopedia of Computer Science and Engineering*, 1–10. doi:10.1002/9780470050118.ecse450

Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal performance*. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge University Press.

Cypher, A. (2010). End user programming on the Web. In *No Code Required* (pp. 3–22). doi:10.1016/B978-0-12-381541-5.00001-8

Donoso, M. (2010). Lista de cotejo para evaluar un software educativo. *Mariadonoso's Blog*. Recuperado el 20 de julio de 2014, de http://mariadonoso.wordpress.com/category/softwares-educativos/lista-de-cotejo-para-evaluar-un-software-educativo/

Dvorak, J. C. (2013). The Hidden Agenda of Code.org. *PC Magazine*. Recuperado el 20 de julio de 2014, de http://www.pcmag.com/article2/0,2817,2428466,00.asp

Erb, B. (2012). *Concurrent Programming for Scalable Web Architectures*. *D-Nb.Info*. Ulm University, Alemania.

Erwin, B., Cyr, M., & Rogers, C. (2000). LEGO engineer and ROBOLAB: Teaching engineering with LabVIEW from kindergarten to graduate school. *International Journal of Engineering Education*, *16*(3), 1–12.

Fowler, A., & Cusack, B. (2011). Enhancing Introductory Programming with Kodu Game Lab: An Exploratory Study. In S. Mann & M. Verhaart (Eds.), *2nd anuual conference of Computing and Information Technology Research and Education New Zealand* (pp. 69–79).

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. *Design* (p. 395). doi:10.1093/carcin/bgs084

Gardner, H. (1999). *Intelligence reframed: Multiple intelligences for the 21st century*. *Intelligence reframed Multiple intelligences for the 21st century* (p. X, 292 S.). doi:10.1177/001698620204600209

Gonzalez, H., & Kuenzi, J. (2012). Science, technology, engineering, and mathematics (STEM) education: A primer.

Grinter, R. E. (1994). Book review: A Small Matter of Programming: Perspectives on End User Computing by Bonnie A. Nardi (MIT Press 1993). *ACM SIGCHI Bulletin*, *26*(4), 80–81. doi:10.1145/191642.1047947

Hale, P., Solomonides, A. E., & Beeson, I. (2012). User-driven modelling: Visualisation and systematic interaction for end-user programming. *Journal of Visual Languages & Computing*, *23*(6), 354–379. doi:10.1016/j.jvlc.2012.08.002

Jameson, D. D. (1844). *Colour-Music*. London: Smith, Elder.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*, 83–137. doi:10.1145/1089733.1089734

Kelleher, C., & Pausch, R. (2006). *Motivating programming: Using storytelling to make computer programming attractive to middle school girls*. *DAI*. Carnegie-Mellon University, Pittsburgh, PA.

Kemeny, J. G., & Kurtz, T. E. (1985). *Back to Basic; The History, Corruption, and Future of the Language*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Koster, R. (2004). *A Theory of Fun*. *Scottsdale Paraglyph* (p. 246). Indianapolis, USA: Paraglyph press.

Lin, J., Wong, J., Nichols, J., Cypher, A., & Lau, T. A. (2009). End-user programming of mashups with vegemite. In *Proceedings of the 14th international conference on Intelligent user interfaces* (pp. 97–106). doi:10.1145/1502650.1502667

Lin, J., Yen, L., Yang, M., & Chen, C. (2005). Teaching computer programming in elementary schools: A pilot study. In *National Educational Computing Conference*.

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, *36*(1), 75. doi:10.1145/1028174.971328

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, *1*(1), 97–123. doi:10.1016/S1045-926X(05)80036-9

Olabe, J. C., Basogain, X., Olabe, M. Á., Maíz, I., & Castaño, C. (2014). Solving math and science problems in the real world with a computational mind. *Journal of New Approaches in Educational Research*, *3*(2), 72–78. doi:10.7821/naer.3.2.75-82

Pardamean, B. (2014). ENHANCEMENT OF CREATIVITY THROUGH LOGO PROGRAMMING. *American Journal of Applied Sciences*, *11*(4), 528–533. doi:10.3844/ajassp.2014.528.533

Pescador, D., & González, C. (2013). Sistemas y redes adaptativas en e-learning. In *Diseño y desarrollo de sistemas y redes adaptativos y colaborativo* (p. 7).

Pilke, E. . (2004). Flow experiences in information technology use. *International Journal of Human-Computer Studies*, *61*(3), 347–357. doi:10.1016/j.ijhcs.2004.01.004

Pratt, T. W., & Zelkowitz, M. V. (2000). *Programming Languages: Design and Implementation*. *Lecture Notes in Computer Science* (Third Edit.). Upper Saddle River, NJ: Prentice Hall.

Repenning, A. (2007). End-User Design. In M. H. Burnett, G. Engels, B. A. Myers, & G. Rothermel (Eds.), *End-User Software Engineering*. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Repenning, A. (2011). Making programming more conversational. In *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011* (pp. 191–194). doi:10.1109/VLHCC.2011.6070398

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch. *Communications of the ACM*, *52*(11), 60. doi:10.1145/1592761.1592779

Rosson, M. B. (2007). Position paper for EUSE 2007 at Dagstuhl. In M. H. Burnett, G. Engels, B. A. Myers, & G. Rothermel (Eds.), *End-User Software Engineering*. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Schwartz, J., Morrison, W., & Witus, D. (2007). *Learn to program with phrogram™!: a guide to learning through game programming using the latest version of kids programming language* (First.). Addison-Wesley Professional.

Soloway, E. (1993). Should we teach students to program? *Communications of the ACM*, *36*(10), 21–24. doi:10.1145/163430.164061

Stolee, K. T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11* (p. 99). New York, New York, USA: ACM Press. doi:10.1145/1953163.1953197

Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, *30*(3), 202–210.

The Logo Foundation. (2011). What is Logo? Recuperado el 20 de julio de 2014, de http://el.media.mit.edu/logo-foundation/logo/index.html

U.S. Department of Labor, B. of L. S. (2014). Computer and Information Research Scientists. *Occupational Outlook Handbook, 2014-15 Edition*. Recuperado el 20 de julio de 2014, de http://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm

Wang, G., Yang, S., & Han, Y. (2009). Mashroom. In *Proceedings of the 18th international conference on World wide web - WWW '09* (p. 861). New York, New York, USA: ACM Press. doi:10.1145/1526709.1526825