NUEVOS MARCOS DE REFERENCIA PARA ESTUDIAR Y EVALUAR EL DESARROLLO DEL PENSAMIENTO COMPUTACIONAL

http://www.eduteka.org/EvaluarPensamientoComputacional.php

Karen Brennan (kbrennan@media.mit.edu)
Mitchel Resnick (mres@media.mit.edu)
MIT Media Lab

Brennan, K., & Resnick, M. (2012). Entrevistas basadas en artefactos para estudiar el desarrollo del Pensamiento Computacional (PC) en el diseño de medios interactivos. Documento presentado en el encuentro anual de la "American Educational Research Association", AERA 2012, Vancouver, BC, Canada.

RESUMEN EJECUTIVO

Pensamiento Computacional es una frase que ha recibido considerable atención durante los últimos años; pero en el momento no existe un acuerdo sobre lo que éste incluye y existe aún menos acuerdo sobre estrategias para evaluar el desarrollo del Pensamiento Computacional en los jóvenes. Nosotros estamos interesados en estudiar las maneras en las que las actividades de aprendizaje basadas en diseño, en especial la programación de medios interactivos, apoya el desarrollo del Pensamiento Computacional en los jóvenes. Durante los años pasados, hemos desarrollado un marco para el Pensamiento Computacional, que emergió de los estudios que hemos realizado sobre las actividades de los diseñadores de medios interactivos. Nuestro contexto es Scratch, entorno de programación que permite a los jóvenes crear sus propias historias, juegos y simulaciones interactivas y, mediante una comunicad en línea, compartir luego esas creaciones con otros jóvenes programadores alrededor del mundo.

La primera parte del documento describe las dimensiones clave de nuestro marco conceptual para el Pensamiento Computacional[1]: ***conceptos computacionales*** (los conceptos de los que deben ocuparse los diseñadores a medida que programan; tales como iteración, paralelismo, etc.); ***prácticas computacionales*** (las prácticas que los diseñadores desarrollan a medida que se ocupan de los conceptos; tales como, depuración de proyectos o remezclas basadas en el trabajo de otros) y ***perspectivas computacionales*** (las perspectivas que los diseñadores construyen sobre el mundo que los rodea y sobre ellos mismos).

[1] http://www.eduteka.org/modulos/9/284/1206/1

La segunda parte del documento describe nuestro enfoque, todavía en evolución, con el que pretendemos evaluar estás dimensiones y que incluye, análisis de portafolio de proyectos, entrevistas basadas en artefactos y diseño de escenarios. Terminamos haciendo una serie de sugerencias para valorar el aprendizaje que se genera cuando los jóvenes se comprometen con la programación.

DISEÑO DE MEDIOS INTERACTIVOS

"Fireflies" (Libélulas) es un video musical creado por Tim, quien tiene 8 años. Él seleccionó una de sus canciones pop favoritas, escuchó cuidadosamente la letra e imaginó cómo ésta se podría representar visualmente. Dibujó los personajes y programó su comportamiento, ensamblando las piezas en una secuencia de tiempo.

"Países" es una simulación creada por Shannon, quien tiene 14 años. A ella le encanta "SimCity" el juego de computador que simula las múltiples dimensiones que tiene una ciudad para que las controle el jugador; ella lo ha jugado durante cientos de horas. Con base en su interés en ese juego y en lo que estaba aprendiendo en ese momento en la clase de historia, desarrolló una simulación de países virtuales en la cual el jugador puede tomar decisiones sobre cómo responder a crisis económicas, agrícolas y políticas.

"10 Niveles" es un juego creado por Renita, quien tiene 10 años, y su hermano menor. Ella interactúa con un juego similar en un sitio de juegos muy popular y decidió diseñar su propia versión de este en la que, en cada nivel, el personaje principal navegue de principio a fin sin chocarse con obstáculos inesperados como puntas afiladas, bolas de fuego y puertas trampa.

Estos tres proyectos los crearon los jóvenes usando Scratch[2], un ambiente computacional de autor (para crear) desarrollado por el grupo de investigación del "Lifelong Kindergaten" del Laboratorio de Medios de MIT. Con Scratch, los jóvenes pueden diseñar sus propios medios interactivos, incluyendo historias, juegos y animaciones, encajando bloques de programación que tienen instrucciones, tal como se encajan los bloques de Lego o las piezas de un rompecabezas (Resnick et al., 2009).

Además del ambiente computacional de autor, existe una comunidad en línea dónde los jóvenes pueden compartir sus proyectos, de la misma forma en que se comparten videos en YouTube. La comunidad en línea de Scratch, que se lanzó en Mayo del 2007 y ha crecido continuamente en los últimos cinco años; cientos de miles de jóvenes creadores, la mayoría entre las edades de 8 a 16 años, han compartido más de 2 millones y medio de proyectos. Los miembros de la comunidad pueden interactuar con los proyectos, ensayarlos o bajarlos para ver cómo funcionan; y, con otros miembros, escribir comentarios o añadirlos como amigos. (Brennan, Resnick, & Monroy-Hernandez, 2010; Brennan, Valverde, Prempeh, Roque, & Chung, 2011).

[2] Eduteka recomienda consultar el módulo "Programación en Educación Escolar" http://www.eduteka.org/modulos/9#Scratch

PENSAMIENTO COMPUTACIONAL

¿Cómo describiríamos lo que Tim, Shannon y Renita están aprendiendo a medida que trabajan con Scratch como diseñadores de medios interactivos? ¿Cuál es el aprendizaje que apoya la programación de medios interactivos si lo comparamos con realizar un video con software de edición o jugar un juego de video?

Nos ha intrigado la frase Pensamiento Computacional[3] como mecanismo para conceptualizar el aprendizaje y el desarrollo que se sucede con Scratch. Aunque, como ya dijimos, el Pensamiento Computacional ha recibido atención considerable en los últimos años no existe acuerdo sobre lo que debe incluir una definición de este (Allan et al., 2010; Barr & Stephenson, 2011; National Academies of Science, 2010). Cuny, Snyder, and Wing (2010) definen el Pensamiento Computacional como los "procesos de pensamiento involucrados en formular problemas y encontrar sus soluciones de manera que las soluciones estén representadas de forma tal que puedan llevarse a cabo efectivamente por un agente que procesa información"; descripción ésta que atinadamente (aunque de forma concisa) encuadra el trabajo de los creadores computacionales.

La frase *Pensamiento Computacional* nos ayuda a pensar sobre el aprendizaje con Scratch; creemos a la vez, que programar con Scratch ofrece un contexto y un conjunto de oportunidades para contribuir en la conversación activa sobre Pensamiento Computacional. Estamos interesados en la forma en que las actividades de aprendizaje basadas en diseño, particularmente, la programación de medios interactivos, apoya el desarrollo del Pensamiento Computacional en los jóvenes. Ese interés lo estimula, en parte, la creciente disponibilidad de herramientas que permiten a los jóvenes diseñar sus propios medios interactivos. Pero, más importante aún, ese interés hunde sus raíces en el compromiso de aprender mediante el diseño de actividades, enfoque constructivista del aprendizaje que resalta la importancia de que los jóvenes se comprometan o involucren con el desarrollo de artefactos externos (Kafai & Resnick, 1996).

Durante los últimos años, estudiando la actividad de la comunidad en línea de Scratch y de los talleres sobre Scratch, hemos desarrollado una definición de *Pensamiento Computacional* que incluye tres dimensiones clave: ***conceptos computacionales*** (los conceptos que emplean los diseñadores a medida que programan); ***prácticas computacionales*** (las prácticas que desarrollan los diseñadores a medida que programan) y ***perspectivas computacionales*** (las perspectivas que los diseñadores construyen sobre el mundo a su alrededor y sobre ellos mismos). Las observaciones y entrevistas fueron fundamentales para ayudarnos a entender el desarrollo longitudinal de los creadores, su participación y sus portafolios de proyectos, durante lapsos de tiempo que comprendieron desde semanas hasta varios años; también fueron cruciales los talleres para entender las prácticas del "creador en acción".

[3] Eduteka recomienda consultar la "Caja de Herramientas para líderes en Pensamiento Computacional" http://www.eduteka.org/modulos/9/284/2062/1

CONCEPTOS DEL PENSAMIENTO COMPUTACIONAL

A medida que los jóvenes diseñan medios interactivos con Scratch, comienzan a interactuar con un conjunto de conceptos computacionales (esquema de los bloques para programar con Scratch), comunes a muchos lenguajes de programación. Hemos identificado siete conceptos que son muy útiles para una amplia gama de proyectos con Scratch y pueden transferirse a otros contextos ya sean estos de programación o no: *secuencias, ciclos, paralelismo, eventos, condicionales, operadores y datos*. Ofrecemos para cada concepto una definición y un ejemplo concreto de un proyecto de Scratch.

Concepto: Secuencias

Un concepto clave en programación, es que una tarea o actividad particular se expresa como una serie de pasos o de instrucciones individuales, que puede ejecutar el computador. Tal como en una receta, una secuencia de instrucciones de programación indica el comportamiento o acción que se debe producir. Por ejemplo, el objeto gato puede programarse para que se mueva una distancia corta a través del escenario y diga, mediante la secuencia de instrucciones que se muestra en la Figura 1, "Yo estoy programando".

FIGURA 1. Ejemplo de una secuencia de instrucciones

Concepto: Ciclos

En el ejemplo anterior, se programó el gato para que se moviera 10 pasos, esperara 0.2 segundos y repitiera la acción, moviéndose otros 10 pasos y esperando otros 0.2 segundos. ¿Qué pasaría si en lugar de una repetición individual de la acción, quisiéramos que el gato se moviera y esperara tres veces más? Fácilmente, podríamos adicionar más bloques *mover* y *esperar*. ¿Pero si lo que queremos ahora es que el gato se mueva y espere 50, 100 o 1000 veces más? Los ciclos son mecanismos que ejecutan la misma secuencia, múltiples veces. La Figura 2 ilustra cómo, de manera más sucinta, se puede usar un ciclo para expresar una secuencia de instrucciones. En lugar de mover y esperar

usando 8 bloques consecutivos, vamos a usar tres bloques: *mover 10 pasos*, seguido por *esperar 0.2 segundos*, anidadas dentro de *repetir* con el número deseado de iteraciones.

FIGURA 2. Secuencia de instrucciones repetidas, expresadas como un ciclo

Concepto: Eventos

Eventos, una cosa que desencadena que otra se suceda; es un componente esencial de los medios interactivos. Por ejemplo, el botón de inicio que activa la reproducción de un video musical o el choque de dos objetos que ocasiona el aumento del puntaje de un juego. La Figura 3 ilustra diferentes situaciones en las que un evento producirá una acción: (1) cuando se presiona la bandera verde el objeto dará vueltas continuamente con incrementos de 15 grados; (2) cuando se presiona la barra espaciadora el objeto se moverá hacia arriba y hacia abajo; y (3), cuando se hace clic sobre el objeto con el ratón, se desplegará durante 2 segundos una burbuja de texto que dice "¡Hola!".

FIGURA 3. Ejemplos de eventos que producen acciones.

Concepto: Paralelismo

Los lenguajes de computador más modernos soportan paralelismo; esto es, secuencias de instrucciones que se suceden simultáneamente. Scratch permite paralelismo entre objetos. Por ejemplo, la escena del baile de una fiesta puede incluir varios personajes que bailan simultáneamente, cada uno con su propia secuencia de instrucciones de baile. Scratch permite también paralelismo dentro del mismo objeto. En la Figura 4, el gato de Scratch se ha programado para realizar, en paralelo, tres conjuntos de actividades como respuesta al evento "*cuando se hace clic en la bandera verde":* (1) suena continuamente una música de fondo; (2) continuamente baila hacia delante y hacia atrás; y (3), se presenta y habla de sus intereses.

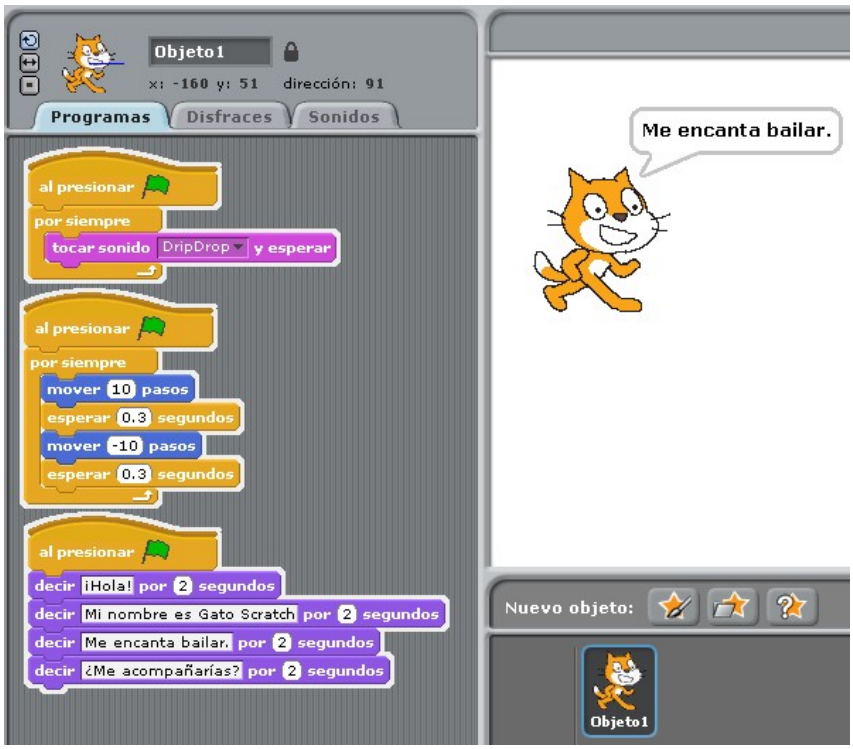

FIGURA 4. Ejemplo de paralelismo con un solo objeto.

Concepto: Condicionales

Otro concepto clave de los medios interactivos son los condicionales; esto es, la habilidad de tomar decisiones con base en ciertas condiciones, que apoyan la expresión de múltiples resultados. La Figura 5 ilustra el uso de un condicional; el bloque *si*, para determinar la visibilidad de un objeto. Si el cubo está *tocando el color amarillo*, debe comenzar a desvanecerse y reaparecer en el próximo nivel del juego; si no lo toca, debe permanecer visible.

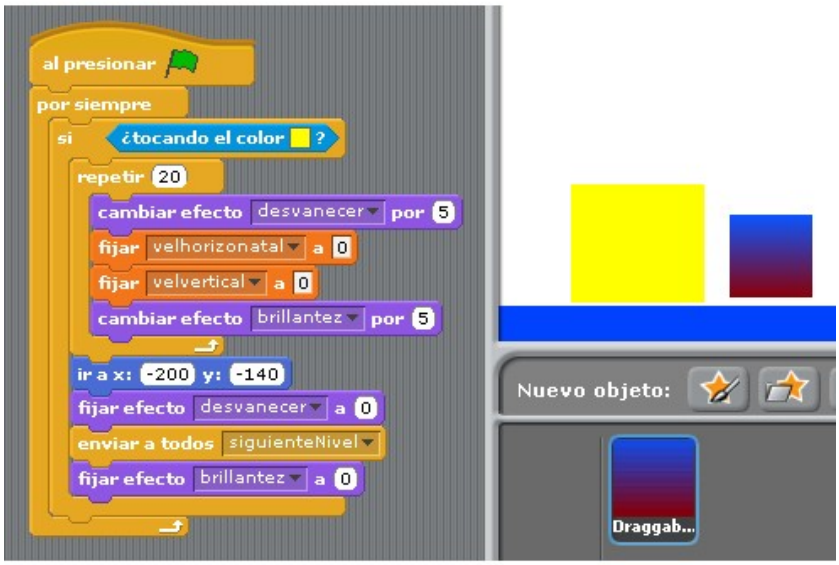

FIGURA 5. Ejemplo de condicionales.

Concepto: Operadores

Los operadores ofrecen apoyo a las expresiones matemáticas, lógicas y de cadenas de caracteres, permitiendo al programador realizar manipulaciones numéricas y de cadenas. Scratch permite un rango de operaciones matemáticas que incluyen suma, resta, multiplicación, división y también, funciones como seno y potencia; además, operaciones con cadenas, incluyendo concatenación y longitud de las cadenas. La Figura 6 ilustra los bloques de operadores de Scratch.

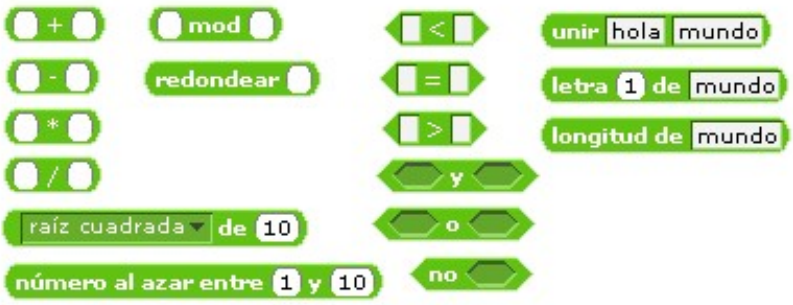

FIGURA 6. Bloques de operadores

Concepto: Datos

Los datos incluyen guardar, recuperar y actualizar valores. Actualmente Scratch ofrece dos contenedores de datos: **variables** que pueden almacenar un solo número o cadena de caracteres y **listas** que pueden contener una colección de números o cadenas. Llevar el puntaje en un juego es, con frecuencia, un motivador para que los jóvenes diseñadores exploren las variables. La Figura 7 demuestra cómo se usa una variable para contabilizar el puntaje en un juego; por cada pescadito que se coma el pescado grande, el puntaje aumenta en 1.

FIGURA 7. Uso de variables para contabilizar puntajes.

PRÁCTICAS DEL PENSAMIENTO COMPUTACIONAL

En nuestras entrevistas y observaciones a los jóvenes diseñadores se hizo evidente que enmarcar el Pensamiento Computacional, solamente en función de los *conceptos,* impedía representar otros elementos del aprendizaje y de la participación de los diseñadores. El paso siguiente en la articulación de nuestro marco para el Pensamiento Computacional, consistió en describir el proceso de construcción, las *prácticas* de diseño que los veíamos emplear a medida que creaban sus proyectos. Las prácticas computacionales se enfocan en el proceso de pensar y de aprender y van más allá de *qué* está usted aprendiendo para centrarse en *cómo* lo está aprendiendo.

Aunque la gente joven que entrevistamos había adoptado una variedad de estrategias y prácticas para desarrollar medios interactivos, observamos cuatro conjuntos de prácticas que se destacan: *ser incremental e iterativo, ensayar y depurar, reusar y remezclar* y, *abstraer y modularizar.* La creación de medios interactivos constituye un contexto poderoso para desarrollar esas prácticas que son útiles en una diversidad de actividades de diseño y no solamente en programación. Para ilustrar estas prácticas en acción, usamos el caso de Renita y su hermano menor y el proceso que siguieron para desarrollar el juego de multiniveles de obstáculo/aventura, que llamaron "10 niveles".

Práctica: Ser incremental e iterativo

Diseñar un proyecto no es un proceso limpio y secuencial en el que primero se identifica un concepto para el proyecto, luego se desarrolla un plan de diseño y luego se implementa el diseño en un código. Es un proceso adaptativo, uno en el que el plan puede cambiar para responder a un enfoque de solución por pequeños pasos. Conversando con los Scratchers (personas que usan Scratch), ellos describen ciclos iterativos de imaginar y construir, desarrollando un poquito, probándolo para luego desarrollar más, basándose en sus experiencias e ideas nuevas. Renita describió este proceso con "10 Niveles" y explicó de qué manera usaba cada iteración como una oportunidad para obtener tanto retroalimentación como nuevas ideas:

> *Entrevistador (E): Bueno, este es un programa complicado. ¿Cuánto tiempo ha estado trabajando en él?*
> Renita (R): Posiblemente dos o tres semanas.
> *E: ¿Trabaja en el todos los días?*
> R: Lo tomo y lo dejo, tal vez lo trabajé durante un mes. Cada que terminaba uno de los niveles, se lo mostraba a mi hermano.
> *E: Usted hablaba de cómo hizo la mayoría del programa y de cómo su hermano ayudó con el concepto del proyecto. ¿Cómo fue el proceso?*
> R: Comenzamos a construirlo sin mucha planeación pero, para los niveles 8, 9 y 10 si hicimos una planeación previa. Mi hermano tenía esa gran idea de que el nivel 10 tuviera pernos y bolas de boliche. El dijo, "Estos deben estar en el nivel 8" yo le respondí, "No en el 10, pues eso es muy difícil" y él contestó "Está bien, está bien, está bien".

Práctica: Ensayar y depurar

Muy rara vez, las cosas funcionan como se habían imaginado; por lo que es crítico para los diseñadores desarrollar estrategias para manejar y anticipar problemas. En las entrevistas, los Scratchers describen las diversas prácticas de ensayo y depuración que han desarrollado, ya sea por ensayo y error, transfiriéndolas de otras actividades o, apoyándose en otras personas más conocedoras. Inicialmente, Renita no recordaba que durante el proceso de desarrollo del proyecto "10 niveles" se hubiera estancado a causa de un problema. Luego, tras unos momentos de reflexión, comenzó a listar varias prácticas de ensayo y depuración que había usado en este y en otros proyectos:

Identificar (la fuente de) el problema
Leer cuidadosamente los programas
Experimentar con los programas
Tratar de escribir de nuevo los programas
Encontrar ejemplos de programas que funcionen
Contarle/preguntarle a alguien acerca del problema
Tomar un descanso

Práctica: Reusar y remezclar

Construir sobre lo que otros ya han hecho, ha sido una práctica de vieja data en programación, que se ha amplificado con las tecnologías de redes que permiten acceder a un amplio rango de trabajos hechos por otros que se pueden reusar y remezclar. Una de las metas de la comunidad en línea de Scratch es apoyar a los jóvenes diseñadores en el reuso y la remezcla, ayudándoles a encontrar ideas y código de programación sobre los que puedan construir, posibilitándoles potenciarse para crear cosas mucho más complejas de las que hubieran podido crear ellos solos. Reusar y remezclar apoya el desarrollo de las capacidades de lectura crítica de código y genera importantes preguntas sobre propiedad y autoría. ¿Qué es razonable tomar prestado de otros? ¿Cómo darles los créditos correspondientes? ¿Cómo evaluar el trabajo colaborativo y cooperativo? El proyecto de Renita se benefició, por lo menos de dos maneras, del reuso y la remezcla. La *idea* del proyecto surgió de la remezcla de un proyecto que ella inicialmente había visto en un sitio Web popular de juegos y más tarde, lo encontró en el sitio Web de Scratch (Figura 8).

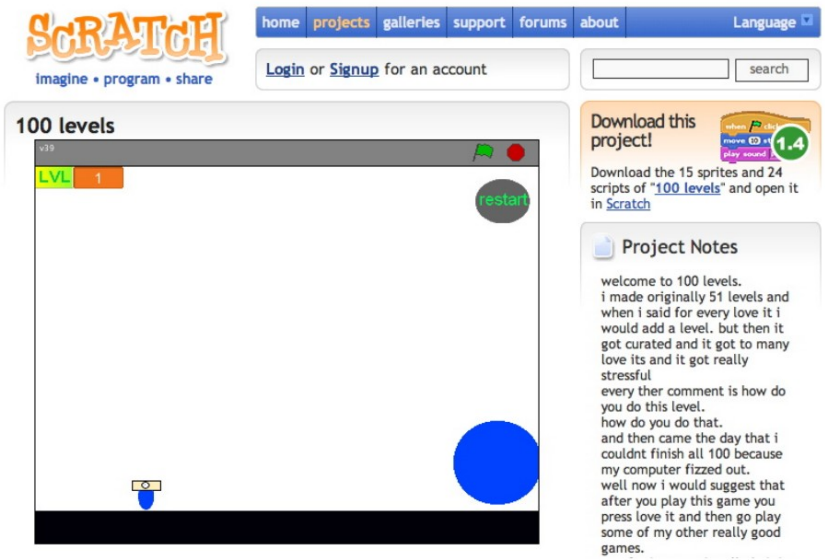

FIGURA 8. Proyecto que sirvió a Renita como inspiración para reusar/remezclar

Ella también remezcló y reusó a nivel del código, incorporando un objeto de la librería de Scratch que incluía el código de simulación "jet-pack", ("JetPack Girl" que se muestra en la Figura 9) y que se convirtió en el personaje principal de su juego.

FIGURA 9. Código reusado de la librería Scratch

Práctica: Abstraer y modularizar

Abstraer y modularizar, lo cual caracterizamos como construir algo grande uniendo colecciones de partes más pequeñas, es una práctica siempre importante para el diseño y la solución de problemas. En Scratch, los diseñadores emplean la abstracción y la modularización en múltiples niveles, desde el trabajo inicial de conceptualizar el problema hasta trasladar el concepto a los objetos individuales y a las pilas de código. La Figura 10 nos muestra una de las formas en las que Renita empleó la modularización y la abstracción en su proyecto "10 niveles", separando los diferentes comportamientos o acciones de su objeto principal, que navega por entre obstáculos. La primera pila

de código (hilo) controla el movimiento del objeto por la pantalla. La segunda pila de código controla la apariencia visual del objeto. La tercera pila de código controla los diversos eventos asociados con los obstáculos tales como, reiniciar el nivel si el objeto colisiona con un obstáculo inesperado. Modularizar el comportamiento del objeto facilita a Renita pensar sobre (y probar/depurar), las diferentes partes y a otras personas, leerlas.

FIGURA 10. Un conjunto complejo de instrucciones organizadas por funcionalidad en tres pilas de código (hilos).

PERSPECTIVAS DEL PENSAMIENTO COMPUTACIONAL

En nuestras conversaciones con los "Scratchers", escuchamos que los jóvenes diseñadores describen como evolucionan en su auto-comprensión, en su relación con otros y respecto al mundo tecnológico que los rodea. Esta fue una dimensión sorprendente y fascinante de su participación con Scratch, una dimensión no captada por nuestra formulación de conceptos y prácticas. De manera que como paso final en la articulación de nuestro marco de referencia para el Pensamiento Computacional, añadimos la dimensión de *perspectivas* para describir los cambios de perspectiva/apreciación que observamos en los jóvenes que trabajan con Scratch.

Perspectiva: Expresar

Las personas están rodeadas de medios interactivos, pero la mayor parte de nuestras experiencias con estos, es como consumidores. Invertimos tiempo en señalar, hacer clic, buscar y "chatear", actividades estas importantes para aprender a usar las Tecnologías de la Información y la

Comunicación (TIC), pero no suficientes para desarrollarnos como pensadores computacionales. Un pensador computacional ve la computación como algo más que un producto de consumo; la computación es algo que ellos pueden usar para diseñar y auto expresarse. Un pensador computacional ve la computación como medio y piensa "Yo puedo crear" y "expresar mis ideas usando este nuevo medio". Ellos lo ven como un medio que es diferente de otras cosas, como lo expresó una niña norteamericana de 13 años:

> Prefiero Scratch a los blogs o sitios de redes sociales como Facebook porque estamos creando juegos interesantes y proyectos que son divertidos de jugar, mirar y descargar. A mí no me gusta simplemente hablar con otras personas en línea; a mí me gusta hablar sobre algo nuevo y creativo.

Y como medio que ofrece bastantes oportunidades, como lo expresó una niña australiana de 9 años:

> *Entrevistador (E): ¿Qué es lo que más le gusta de Scratch?*
> Scratcher (S): Pues, tal vez que cuando usted sube todo lo que ha trabajado tiene un proyecto. O tal vez, simplemente es la creatividad que posibilita Scratch.
> *E: ¿Qué quiere decir con eso? ¿Puede ampliarnos un poco lo que significa ser creativo con Scratch?*
> S: Pues, es que simplemente las posibilidades son ilimitadas. No es solo que usted puede hacer este proyecto o aquel y que eso sea todo lo que usted puede hacer.

Perspectiva: Conectar

La creatividad y el aprendizaje son prácticas profundamente sociales por lo que no sorprende que diseñar medios computacionales con Scratch se enriquezca mucho mediante la interacción con otros. En entrevistas y observaciones, notamos la amplia gama de formas en las que la práctica creativa de un determinado "Scratcher" se beneficia cuando puede acceder a otros, en encuentros cara a cara o, particularmente, en el caso de Scratch, mediante la comunidad en línea, las redes en línea (Brennan & Resnick, en prensa). Los jóvenes describen el poder que tiene la posibilidad de acceder a nuevas personas, proyectos y perspectivas por medio de estas redes; un cambio de perspectiva expresado sucintamente así: "Yo puedo hacer cosas diferentes cuando tengo acceso a otros".

En entrevistas, el poder establecer contacto con otros se describió de dos maneras: el valor de crear *con* otros y el valor de crear *para* otros. En referencia a crear *con* otros, los jóvenes Scratchers describieron cómo fueron capaces de hacer más de lo que ellos hubieran podido hacer individualmente, ya fuera por las respuestas que recibieron a sus preguntas en foros en línea, tales como ayuda en la corrección de un problema particular de un proyecto; o por el estudio y la remezcla del código de otros, tales como la búsqueda de una base de proyecto de desplazamiento lateral para construir a partir de él; o estableciendo alianzas y colaboraciones intencionadas, como Scratchers que conforman "estudios de diseño" o "compañías de producción" para crear proyectos en conjunto. Respecto a crear *para* otros, los jóvenes Scratchers experimentan el valor de tener una audiencia

auténtica. Aprecian el que otros estuvieran comprometidos con sus creaciones y las apreciaran; así fuera *entreteniendo* a otros, generando una audiencia de seguidores para una serie de proyectos por entregas (tipo novela), *comprometiendo* a otros, como en el caso del diseño de una encuesta para que la respondan otros miembros de la comunidad, *ofreciendo suministros* a otros mediante el desarrollo de componentes para que otros los utilicen en sus propios proyectos, o *educando* a otros, realizando proyectos de tutoriales que ayudan a otros Scratchers a aprender sobre Scratch algo que desconocían; por ejemplo, de qué manera usar la trigonometría en simulaciones de física o cómo hacer proyectos que tengan amplia aceptación.

Perspectiva: Preguntar

Como dijo Bandura (2001), "la vida diaria cada vez está más regulada por tecnologías complejas que la mayoría de las personas no entienden ni creen que puedan hacer algo para influir en ellas" (p. 17). Con la perspectiva computacional de preguntar, buscamos indicadores de que los jóvenes no sientan esa desconexión entre las tecnologías que los rodean y sus habilidades para negociar las realidades del mundo tecnológico. Los jóvenes deben sentirse empoderados para formular preguntas sobre y con tecnología (TIC); "Puedo usar la computación para hacer preguntas que le den sentido a cosas computacionales en el mundo". Como ejemplo de este cambio en la visión del mundo, un niño de 11 años describe la nueva perspectiva con la que ve los objetos que lo rodean:

> Me encanta Scratch. Espera, déjame refrasear lo anterior, Scratch es mi vida. He hecho muchos proyectos. Ahora yo tengo lo que se llama "mente de programador"; esto es, la forma en que pienso sobre cómo cualquier cosa está programada. Lo anterior ha incluido desde tostadoras, sistemas eléctricos de carros y muchas cosas más.

Preguntar incluye interrogar sobre lo que se da por sentado y, en algunos casos, responder a ese cuestionamiento mediante diseño. Por ejemplo, el entorno de programación Scratch es un artefacto computacional que por su diseño tiene posibilidades y limitaciones. Algunos miembros jóvenes de la comunidad cuestionan esas limitaciones y conforman grupos para realizar una versión derivada de Scratch que contiene bloques que ellos piensan que este debe incluir y además, desarrollan un sitio Web donde otras personas pueden descargar esa versión modificada de Scratch. Esto implica no solo reconocer que Scratch es un artefacto diseñado en el mundo que se puede modificar; sino también, que ellos como diseñadores de medios computacionales, están empoderados para modificarlo.

EVALUAR EL APRENDIZAJE MEDIANTE EL DISEÑO

Habiendo articulado nuestro marco de referencia para el Pensamiento Computacional (conceptos, prácticas y perspectivas), describiremos ahora tres enfoques para evaluar el desarrollo de este pensamiento en jóvenes comprometidos en el diseño de actividades con Scratch. Para cada uno de ellos, describiremos los detalles del proceso de evaluación, cuando estén disponibles ofreceremos ejemplo de datos y discutiremos las fortalezas y limitaciones de cada enfoque.

***ENFOQUE # 1:* Análisis del portafolio de proyectos**

Cada uno de los miembros de la comunidad en línea de Scratch tiene una página de perfil en la que aparecen sus creaciones, así como otras dimensiones de su participación, tales como proyectos que ha marcado como favoritos y los Scratchers que sigue. Por ejemplo, en la Figura 11 se muestra la página con el perfil de un Scratcher de 17 años que ha sido miembro de la comunidad por más de tres años y ha publicado 49 proyectos.

FIGURA 11. Página de perfil de miembro de la comunidad Scratch.

Investigadores en un "College" de New Jersey (USA), desarrollaron un conjunto de visualizadores llamado "Scrape"[4] (Raspar) que analiza los bloques de programación al interior de los proyectos de Scratch (Wolz, Hallberg, & Taylor, 2011). Nuestro primer enfoque para evaluar el desarrollo del Pensamiento Computacional utiliza una de las herramientas de "Scrape" (el "User Analysys" - la herramienta análisis del usuario) para analizar el portafolio de proyectos que ha subido un miembro particular de la comunidad y generar una representación visual de los bloques que ha usado u omitido, en cada proyecto.

La visualización del trabajo del miembro de la comunidad al que se refiere la Figura 11, se muestra en la Figura 12. Cada columna representa un proyecto y todos los bloques que este contiene, mientras cada fila representa un bloque específico de Scratch. Una sombra oscura indica el uso más

[4] http://happyanalyzing.com/

frecuente de un bloque en el proyecto. La columna final identifica los bloques que nunca se han usado.

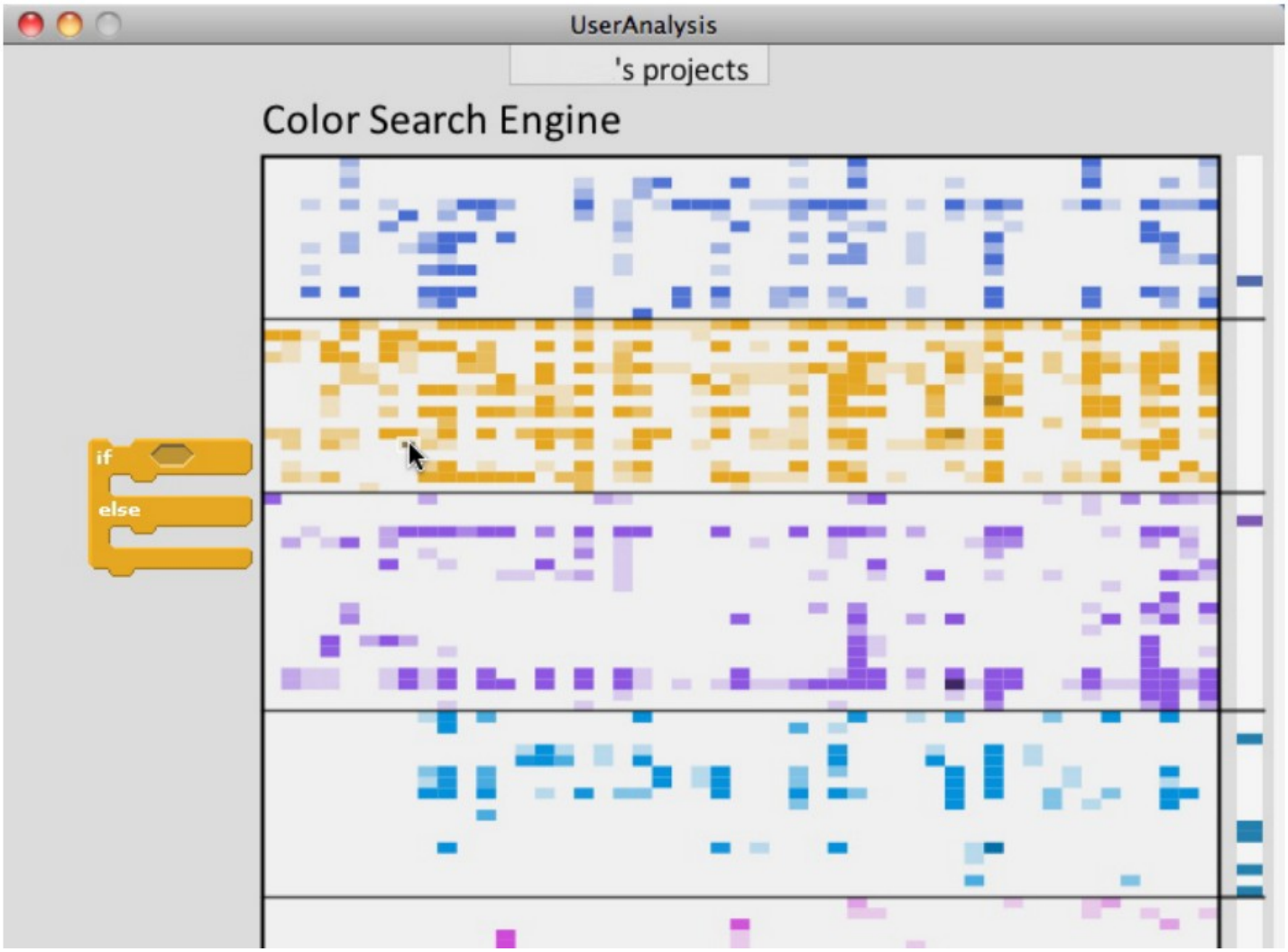

FIGURA 12. Análisis del uso de bloques de programación por un Scratcher experimentado

En comparación, la Figura 13 muestra la visualización con la herramienta "User Analysis visualization" de "Scrape" de un Scratcher novato. Este miembro de 11 años, lleva una semana de membresía y ha creado 19 proyectos. Pero en contraste con el miembro representado en la Figura 12, no ha tenido experiencias con la mayoría de los bloques de Scratch.

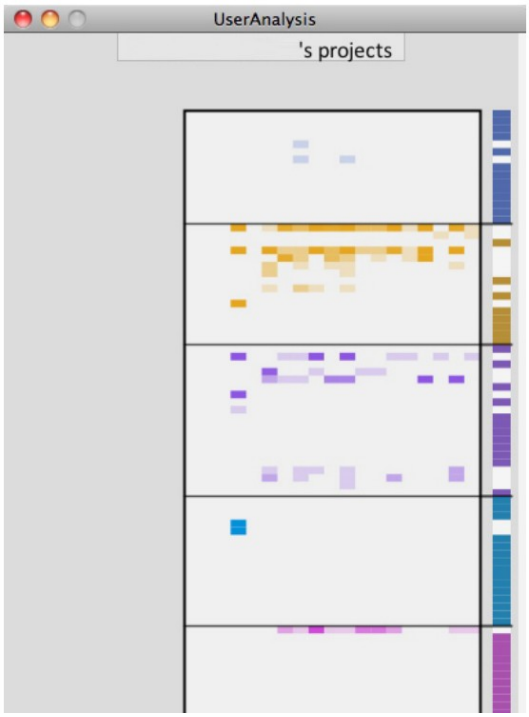

FIGURA 13. Análisis del uso de bloques de programación por un Scratcher novato

Fortalezas

Como se describió en la primera parte de este documento, nuestro marco de referencia mapea bloques específicos de Scratch. Por esto, el enfoque del "User Analysis" para analizar los bloques de un proyecto ofrece un registro de conceptos computacionales con los que se ha encontrado el Scratcher. La naturaleza formativa de esta evaluación tiene para nosotros especial atractivo. La herramienta "User Analysis"[5] se enfoca en un conjunto de trabajos desarrollados en el tiempo, enfatizando así la naturaleza evolutiva y de desarrollo del portafolio del Scratcher en lugar de, por ejemplo, la valoración sumativa de un proyecto final único.

Limitaciones

Nuestro enfoque inicial de analizar el contenido del proyecto como medio para evaluar el Pensamiento Computacional, reveló muy pronto sus limitaciones. Este enfoque está totalmente orientado al producto y no hace revelación alguna sobre el *proceso* que se sigue al realizar proyectos; como tampoco dice nada, sobre las *prácticas* particulares del Pensamiento Computacional que se pudieron haber empleado. Esta falta de información sobre el proceso impacta la evaluación de los conceptos, pues desconoce si el creador estaba en capacidad de hacerlo por sí mismo, lo que se contrapone con recibir ayuda de otras personas o proyectos. Además, del nivel de su comprensión sobre los conceptos asociados con bloques particulares. Lo anterior, también en contraposición a conceptos a los que ellos habían estado ya "expuestos".

[5] http://happyanalyzing.com/

Aprendimos con las entrevistas y observaciones que muchos jóvenes no publican la totalidad de sus proyectos en la comunidad en línea de Scratch. En especial, proyectos en construcción o abandonados que no se publicaban o se subían a una cuenta alternativa de prueba. Este tipo de proyectos pueden ser particularmente interesantes desde la perspectiva del desarrollo, pues en ellos pueden sobresalir áreas retadoras o en las que hay confusión conceptual.

Finalmente, este enfoque se centra en los bloques que aparecen en los proyectos, pero existen otras formas menos automatizadas de estudiar los perfiles de los miembros que conforman la comunidad Scratch. Por ejemplo, se pueden analizar los tipos de proyectos elaborados. ¿Todos los proyectos son del mismo tipo? Esto es, ¿todos son juegos o todos son historias? ¿Demuestra el creador habilidad para desarrollar diferentes tipos de proyectos con este medio? Más allá de los proyectos, el análisis puede extenderse a estudiar los comentarios de los Scratchers. ¿Qué dice el creador sobre su propio trabajo? ¿Qué dice cuando responde a los proyectos de otros? ¿Hasta qué punto se establecen conexiones con otros creadores?

ENFOQUE # 2: Entrevistas basadas en artefactos

Estudiar los portafolios en línea de los proyectos de los Scratchers genera numerosos interrogantes, interrogantes que pensamos pueden explorarse mejor en conversaciones directas con ellos. Nuestro segundo enfoque para evaluar el desarrollo del Pensamiento Computacional consistió en realizar entrevistas sustentadas en artefactos. En el curso de un año entrevistamos 31 Scratchers, con un rango de edad entre 8 y 17 años; ubicados geográficamente en Norte América, Europa y Asia; que hubieran participado entre 1 mes y 4 años; cuya sofisticación técnica/estética estuviera en un rango entre principiantes y expertos; y que el 40% de ellos fueran mujeres, pues esto refleja la participación dentro de la comunidad en línea. La mayoría de estos Scratchers se seleccionaron por un muestreo al azahar, pero otros se escogieron después de que respondieron a una invitación amplia que se hizo a la comunidad.

El tiempo dedicado a las entrevistas estuvo entre 60 y 120 minutos. El protocolo de entrevista se organizó en cuatro secciones principales:

1- Antecedentes
   a. Introducción a Scratch: *¿Cómo conoció usted acerca de Scratch? ¿Qué es Scratch?*
   b. Prácticas actuales: *¿Dónde usa usted Scratch? ¿Qué hace con él? ¿Le ayudan otras personas? ¿Ayuda usted a otras personas?*
2- Creación del proyecto
   a. Marco del proyecto: *¿Cómo o de dónde surge la idea de su proyecto?*
   b. Proceso del proyecto: *¿Cómo empieza a hacer el proyecto? ¿Qué pasa cuando se atasca?*
3- Comunidad en línea
   a. Presentación a la comunidad en línea: *¿Qué hace usted en la comunidad en línea? ¿En qué consiste la comunidad en línea de Scratch?*

b. Otras personas, otros proyectos: *¿Cómo encuentra personas y proyectos interesantes? ¿Cómo interactúa con otros Scratchers?*

4- Mirando hacia delante
   a. Scratch: *¿Qué le gusta o le disgusta de Scratch? ¿Qué le conservaría, adicionaría o cambiaría?*
   b. Tecnología: *¿Qué otras cosas tecnológicas le gusta hacer?*
   c. Más allá de la Tecnología: *¿Qué otras cosas no relacionadas con la tecnología que le gusta hacer?*

La Sección 2, Creación del proyecto, fue la que tuvo más significado para evaluar los conceptos y prácticas del Pensamiento Computacional. Pedimos a los entrevistados escoger dos proyectos que encontraran interesantes para discutir. Para cada proyecto, comenzamos preguntando sobre la historia y motivación para realizarlo. Luego lo corrimos para ver si funcionaba. Solicitamos al creador relatar el proceso de desarrollo del proyecto: ¿cómo lo inició, como evolucionó este durante su desarrollo, qué conocimientos para poder realizar el proyecto eran importantes para ellos, qué problemas encontraron a lo largo del proceso y cómo manejaron esos problemas? Para terminar, les solicitamos hacer algunas reflexiones sobre el artefacto, tales como de qué estaban más orgullosos, qué quisieran cambiar y qué los sorprendió. Con este enfoque logramos conversaciones detalladas con los Scratchers sobre elementos particulares de programación en un proyecto; por ejemplo, preguntarles cómo funciona una pila de código o por qué utilizaron un bloque particular. Además de llevar a cabo descripciones enriquecidas de sus prácticas de desarrollo.

Este enfoque sacó a la luz una debilidad de nuestro anterior enfoque de analizar el portafolio de proyectos en base al análisis de bloques. Consideremos el análisis de un Scratcher de 13 años que ha venido usando Scratch durante 3 años y medio y ha creado 163 proyectos (Figura 14). Desde la visualización, veíamos que este Scratcher había desarrollado numerosos proyectos y experimentado con una diversidad de bloques.

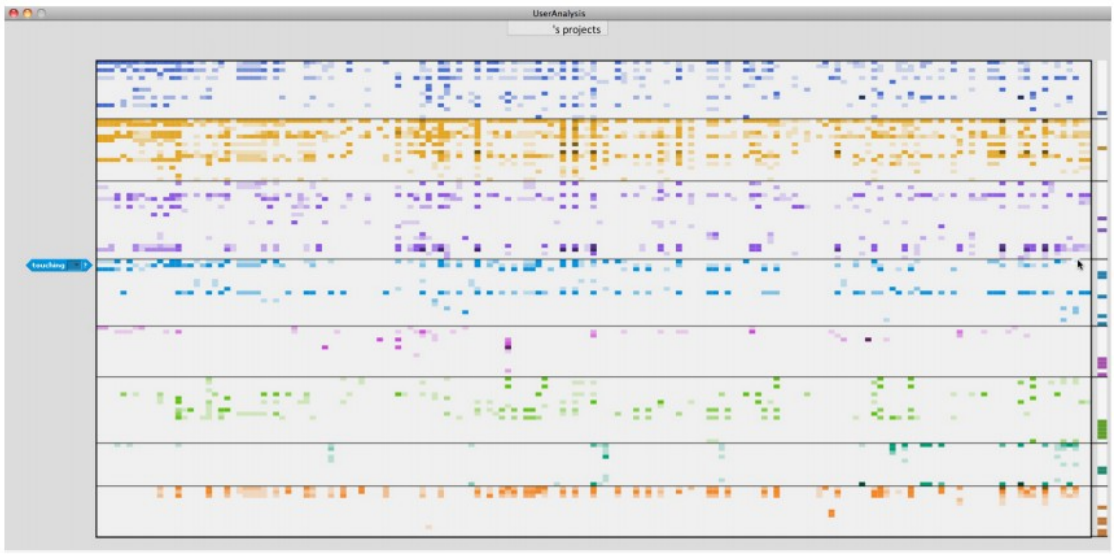

FIGURA 14. Análisis del uso de bloques de programación por un Scratcher aparentemente competente

Pero en la entrevista se hizo evidente, que a pesar de la aparente fluidez del Scratcher, existían brechas conceptuales significativas. Por ejemplo, uno de los proyectos seleccionados era una gráfica, dibujada en tiempo real, que es proporcional a la altura del sonido medida por el micrófono del computador; mientras más alto el sonido, mayor es el pico de la gráfica (Figura 15).

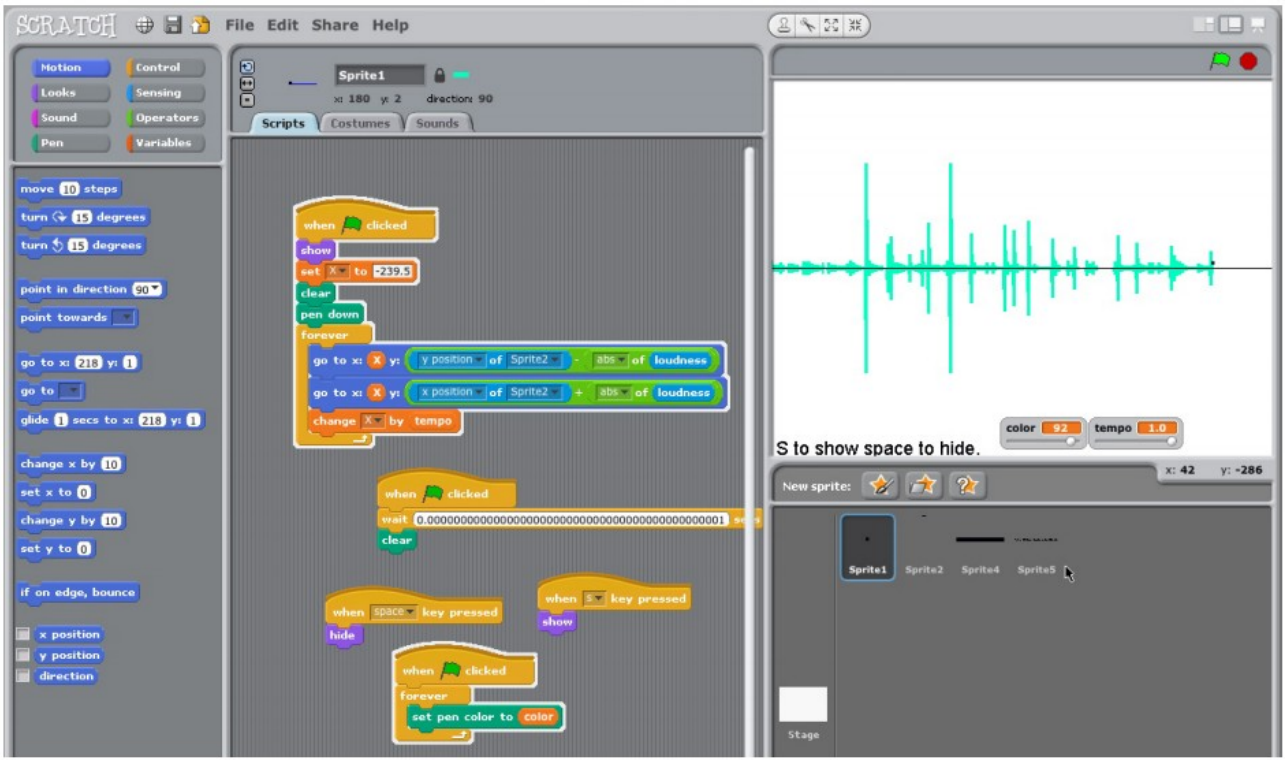

FIGURA 15. Proyecto creado por el Scratcher e ilustrado en la figura 14

A medida que hablábamos respecto al proyecto y sobre su funcionamiento, uno de los entrevistadores quiso saber más sobre una sección particular del código (Figura 16) y preguntó, "¿Cómo funciona esto?". El Scratcher no pudo explicar ni una parte de este. Comentó que había visto un proyecto similar en el sitio Web, lo había bajado para ver el código y había comenzado a arrastrar bloques que encajaban hasta que de alguna manera le funcionó.

FIGURA 16. Código reutilizado por un Scratcher que no lo ha entendido completamente

Fortalezas

El intercambio con ese Scratcher nos sirvió para recordar, de manera perentoria, que la presencia de un elemento de código dentro de un proyecto no necesariamente indica que el diseñador tiene una comprensión profunda del elemento de código y reforzó además, la fortaleza de llevar a cabo entrevistas basadas en artefactos, para desarrollar una mejor comprensión de la fluidez del Scratcher sobre conceptos específicos. Los conceptos ya no están simplemente *presentes* o *ausentes*, como en el primer enfoque; emerge una caracterización más pormenorizada de la ubicación que tienen los Scratchers en diferentes puntos de su trayectoria hacia la comprensión. Por ejemplo, consideremos un Scratcher que puede explicar qué es un concepto o un bloque particular, pero es incapaz de usarlo significativamente dentro de un contexto. O un Scratcher que puede leer el código de otro y explicar cómo funciona, pero al trabajar de manera independiente, tiene dudas acerca de cómo seleccionar y utilizar el mismo concepto en un nuevo contexto.

El formato diligenciado en base a la discusión permitió expandirse, no enfocarse solo en el producto, sino incluir también el proceso. Lo anterior nos ofreció oportunidades para evaluar cómo estaban empleando los jóvenes prácticas del Pensamiento Computacional a medida que desarrollaban los proyectos. Con las prácticas analizamos varios indicadores de fluidez: ¿Eran conscientes de y capaces de articular sus prácticas de diseño y sus estrategias? ¿Tenían en su repertorio un rango de estrategias? ¿Las estrategias que desarrollaron les ayudaban a alcanzar sus metas de diseño?

Limitaciones

El enfoque de entrevista basada en artefactos, demanda bastante tiempo pues requiere como mínimo una hora con el Scratcher. Se adiciona a lo anterior, que idealmente las entrevistas deberían repetirse en el tiempo y en muchos momentos, para obtener un retrato del desarrollo.

Aunque pudimos discutir el proceso de desarrollo de algunos proyectos particulares, esta deliberación se veía limitada por lo que el Scratcher estaba en capacidad de recordar y por lo general no se exploraban las prácticas en tiempo real. Cuando se pedía a algunos Scratchers que describieran una situación en la que se habían "estancado", durante el desarrollo del proyecto, inmediatamente respondían "Yo nunca me estanqué!" Para algunos esta primera respuesta rápida era el reflejo de una limitación en la memoria, mientras que para otros reflejaba el deseo de comunicar experticia o dominio.

Esta discusión del proceso también se veía constreñida por los dos proyectos de Scratch seleccionados para discutir. Al preguntar por qué habían seleccionado un proyecto, muchos Scratchers respondían que era particularmente "llamativo" o "chévere" o "popular"; era un proyecto del que estaban particularmente orgullosos. Por supuesto, estos proyectos tenían retos o dificultades, pero estos invariablemente se habían superado y no eran aspectos con los que estuvieran luchando activamente. Hubo algunos casos en los que el Scratcher entrevistado solicitó ayuda con sus proyectos y hasta en una de las entrevistas, el Scratcher comenzó dando a los entrevistadores una lista de conceptos relacionados con Scratch para que se los explicaran.

ENFOQUE # 3: Diseño de escenarios

Nuestro tercer enfoque de evaluación consistió en diseñar escenarios. Estos escenarios se realizaron en colaboración con investigadores del "Education Development Center (EDC)" – Centro para el desarrollo educativo - como parte de un patrocinio de la "National Science Foundation – NSF". Este centro se dedica al desarrollo del Pensamiento Computacional mediante la realización de actividades de programación en las que se usa Scratch. A diferencia de los otros enfoques de evaluación descritos en este documento, este diseño de escenarios se aplicó exclusivamente en ambientes de aula. Investigadores de la EDC ensayaron esta herramienta de evaluación con un pequeño grupo de estudiantes en varias instituciones educativas y en diferentes asignaturas y grados escolares.

Desarrollamos tres conjuntos de proyectos de Scratch de complejidad incremental. En cada conjunto, había dos proyectos que demandaban los mismos conceptos y prácticas, pero se valían de diferentes estéticas para atraer distintos intereses. En una serie de tres entrevistas, se presentaba a los estudiantes el diseño de escenarios y se les daba a entender que eran proyectos creados por otro joven Scratcher. Luego, se les pedía seleccionar un proyecto de cada conjunto y (1) explicar qué hace el proyecto seleccionado; (2) describir cómo puede extenderse; (3) arreglarle una falla; y (4) remezclar el proyecto adicionándole alguna característica. La combinación de estas cuatro actividades emergió de varias actividades independientes (presentaciones, críticas, depuración, retos y remezclas), con las que habíamos venido experimentando en talleres tanto para jóvenes Scratchers como para educadores. Los tipos de proyectos y la secuencia de conceptos seguía el marco de referencia trazado por La Guía Curricular de Scratch (Brennan, 2011). Estos proyectos pueden descargarse de: http://bit.ly/ScratchDesignScenarios

Conjunto #1: Nombre y Actuación

En el proyecto *Nombre*, Dean, su creador, había diseñado una animación que despliega su nombre. ¿Cómo podría extenderse este proyecto? Dean quería que la N apareciera después de la A y no al mismo tiempo. ¿Cuál es la falla? ¿Cómo se arregla? Dean quería que la N hiciera algo interesante, tal como lo hacían las otras letras, pero solo cuando se presionaba esa letra. ¿Cómo adicionar esta característica?

En el proyecto de *Actuación,* Kelly había diseñado la animación de una actuación. ¿Cómo se podría extenderse ese proyecto? Kelly quería que el intérprete cantara mientras se movía y no después de hacerlo. ¿Cuál es el problema? ¿Cómo lo arreglamos? Kelly quería además que cada tambor comenzara a sonar solo cuando se le hiciera clic. ¿Cómo adicionar esta característica?

Estos dos proyectos (Figura 17) ejemplifican los conceptos de secuencia, ciclos, paralelismo y eventos del Pensamiento Computacional.

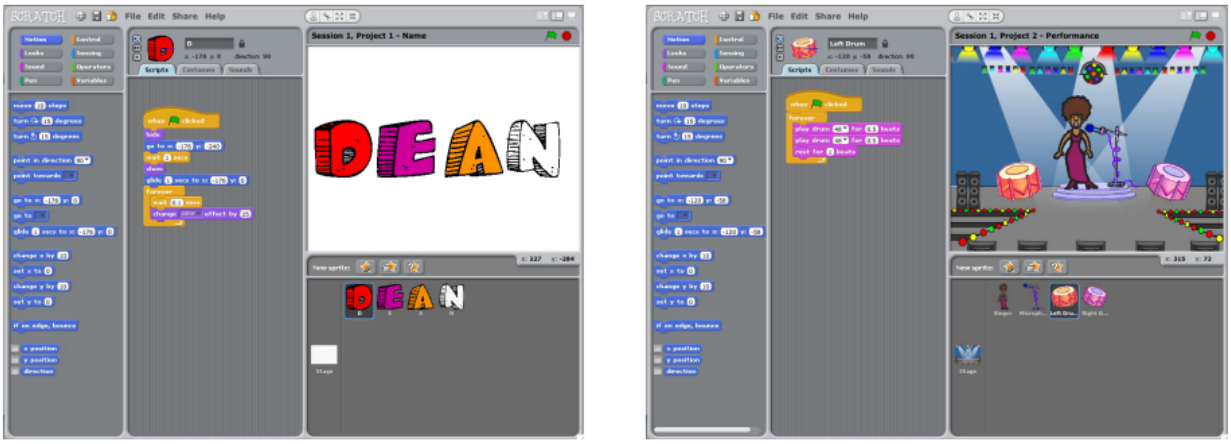

Figura 17. Proyectos Nombre y Actuación.

Conjunto #2: Conversaciones bajo el agua y escenas deportivas

En el proyecto *Conversaciones bajo el agua*, Miguel había diseñado un proyecto que mostraba una conversación entre una ballena y un pulpo. ¿Cómo podría extenderse este proyecto? Miguel quiere que la ballena diga, "No mucho" después de que le pulpo diga "Hola ballena ¿qué hay de nuevo?" ¿Cuál es el problema? ¿Cómo lo arreglamos? También quiere que el cangrejo aparezca después de que la ballena diga "No mucho" ¿Cómo adicionar esta característica?

En el proyecto de *Escenas deportivas,* Gracie había diseñado un proyecto que le ayudaba a las personas a aprender sobre deportes. ¿Cómo podría extenderse este proyecto? Gracie quiere mostrar el fondo del Tenis cuando el botón "Tenis" se presiona. ¿Cuál es el problema? ¿Cómo lo arreglamos? También desea que al presionar el botón "Beisbol", la bola de beisbol aparezca y diga" ¿sabe usted quién inventó el beisbol?" ¿Cómo agregar esta característica?

Estos dos proyectos (Figura 18) ejemplifican los conceptos de secuencia, ciclos, paralelismo y eventos del Pensamiento Computacional.

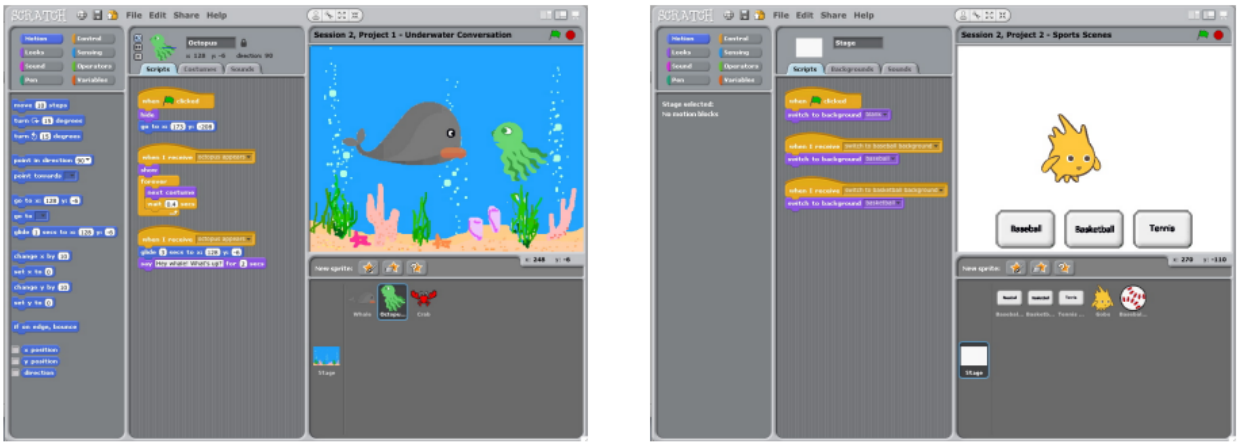

Figura 18. Proyectos Conversaciones bajo el agua y Escenas deportivas

Conjunto #3: Salto y Examen de frutas

En el proyecto de *salto*, Amaya diseñó un triple nivel de obstáculos para un juego de salto. ¿Cómo podría extenderse este proyecto? Quiere además que el juego se detenga cuando se completen los tres niveles. ¿Cuál es el problema? ¿Cómo lo arreglamos? Amaya desea agregar otro nivel a su juego de salto ¿Cómo lo adicionamos?

En el proyecto de *Examen de frutas,* Mylo diseñó una prueba con una manzana, un banano y una naranja. ¿Cómo podría extenderse este proyecto? Él quiere que aparezca la palabra "Perfecto" al final del proyecto, si se identifican correctamente las tres frutas; y "Casi" si esto no sucede. Pero el proyecto siempre muestra la palabra "Perfecto". ¿Cuál es el problema? ¿Cómo lo arreglamos? Mylo quiere adicionar otra fruta a la prueba. ¿Cómo la agregamos?

Estos dos proyectos (Figura 19) ejemplifican los conceptos de secuencia, ciclos, paralelismo, eventos, condicionales, operadores y datos del Pensamiento Computacional.

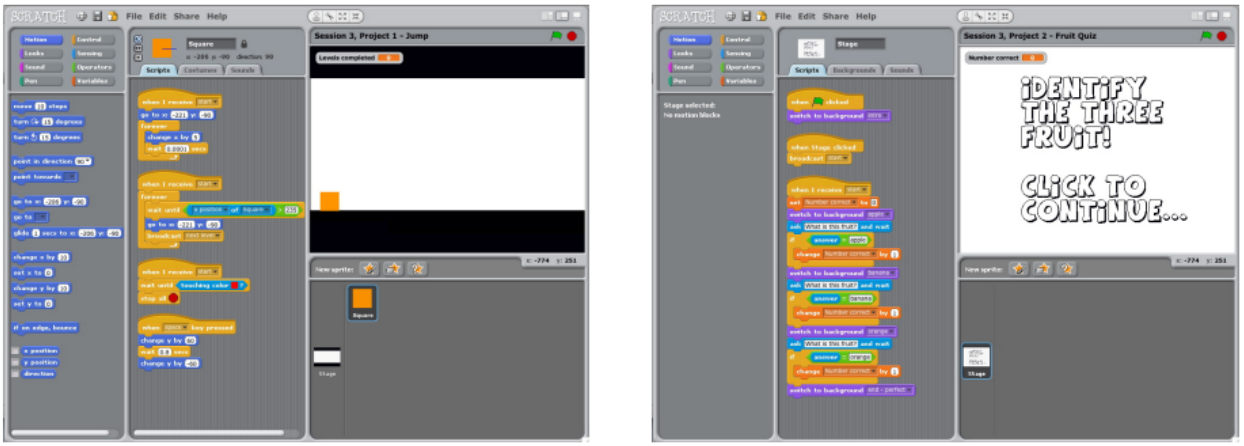

Figura 19. Proyectos Salto y Examen de frutas.

Fortalezas

Como enfoque de evaluación, el diseño de escenarios tiene tres grandes fortalezas. La primera; los escenarios, con los cuatro conjuntos de preguntas que se aplican a cada uno de los proyectos, ofrecen la oportunidad de explorar sistemáticamente diferentes formas tanto de conocer (tales como criticar, extender, resolver problemas y remezclar); como de constatar la fluidez con diferentes conceptos y prácticas. La segunda; se pensó usar el diseño de escenarios en tres momentos a lo largo del tiempo, enfoque este que favorece una mirada formativa o de desarrollo; y finalmente, los escenarios hacen hincapié en el proceso en acción, más que en el proceso vía memoria. En las entrevistas basadas en artefactos, se pedía a los Scratchers describir una situación en la que ellos identificaron y resolvieron (o no) un problema con el software. Lo anterior, recurriendo a sus recuerdos o reflexiones sobre la experiencia. En el diseño de escenarios por ejemplo, los entrevistadores, evaluadores o docentes, pueden observar directamente las prácticas de diseño y las estrategias de solución de problemas de los Scratchers.

Limitaciones

Tal como ocurre en las entrevistas basadas en artefactos, el diseño de escenarios también demanda un tiempo considerable. Especialmente en la solución de problemas y en la extensión de las actividades. ¿Qué tanto tiempo debe darse para trabajar en el desarrollo de actividades? ¿Se debe ayudar si la persona, de manera consciente o no, se atasca o sigue un camino improductivo?

Aún más, la naturaleza de las preguntas y el uso de proyectos seleccionados externamente puede no conectarse con los intereses y la motivación intrínseca de los aprendices. Como el diseño de escenarios se presenta como una manera de ayudar a otro Scratcher, para algunos, los sentimientos de ayudar a alguien más, en especial si no logran ofrecerle una respuesta, pueden transformarse en sentimientos de juicio o de prueba.

CÓMO CONECTAR EL MARCO CONCEPTUAL Y LA EVALUACIÓN

Volviendo a nuestro marco conceptual para el Pensamiento Computacional ¿de qué manera pueden estos diferentes enfoques apoyar la valoración de **conceptos, prácticas y perspectivas** computacionales? La Tabla No 1, resume las fortalezas y limitaciones de cada uno de los enfoques descritos en la sección anterior.

Tabla No 1. Fortalezas y limitaciones de los enfoques de evaluación

	Conceptos	**Practicas**	**Perspectivas**
Enfoque # 1: *Análisis de Proyecto*	La presencia de bloques indica coincidencias conceptuales	N/A	N/A (posiblemente extendiendo el análisis para incluir otros datos del sitio web, tales como comentarios)
Enfoque # 2: *Entrevistas con base en artefactos*	Aspectos de la comprensión de conceptos, pero con un grupo limitado de proyectos	Si, con base en experiencias propias, de diseño auténtico, pero sujeto a limitaciones de memoria	Posible, pero difícil de preguntar directamente
Enfoque # 3: *Diseño de Escenarios*	Proyectos con matices y variedad en la comprensión conceptual, pero seleccionados externamente	Si, en tiempo real y en una nueva situación, pero con proyectos seleccionados externamente	Posible, pero difícil de preguntar directamente

En general, sentimos que el avance logrado a partir del primer enfoque y hasta llegar al tercero, fue productivo; especialmente, porque nos condujo a la comprensión de un mayor número de aspectos (matices) respecto a la fluidez de los Scratchers con los conceptos computacionales y nos permitió acceder a datos más valiosos sobre las prácticas computacionales de estos. Ninguno de estos tres enfoques fue particularmente efectivo para entender los cambios en las perspectivas del pensamiento computacional. Es un reto preguntar explícitamente a un Scratcher de qué manera participar en una actividad como programar con Scratch, ha contribuido a que haga un cambio en la comprensión de sí mismo o del mundo. Los descubrimientos sociales y psicológicos, por lo general, emergen de manera casual mediante entrevistas con un Scratcher o con otros cercanos a este, tales como padres o maestros; lo opuesto al cuestionamiento directo.

Dado que ninguno de los enfoques probó ser suficiente, una combinación de estos puede ser apropiada. Sin embargo, entendemos que ciertas limitaciones como disponibilidad de tiempo para realizar la evaluación o el número de aprendices, hace esto difícil si no imposible.

SEIS SUGERENCIAS PARA EVALUAR EL PENSAMIENTO COMPUTACIONAL MEDIANTE LA PROGRAMACIÓN

En general, necesitamos reflexionar sobre cómo desarrollarse como pensador computacional en diferentes contextos, escalas de tiempo, con diferentes motivaciones y con diferentes estructuras y ayudas; y, de qué manera, esas diferencias conducen a distintos enfoques de evaluación. A pesar de las variaciones en los entornos de aprendizaje, que pueden inclusive no incluir el pensamiento computacional como marco de referencia explícito para el aprendizaje, terminamos este documento con un conjunto de sugerencias generales para la evaluación del aprendizaje que se da cuando los jóvenes se interesan en la programación, que argumentamos, constituyen un marco o escenario valioso para desarrollar capacidades para el Pensamiento Computacional.

SUGERENCIA #1: Apoyar el aprendizaje futuro

Creemos que las mejores formas de evaluación son aquellas útiles para los aprendices. Los tres enfoques para evaluar descritos en este documento solo se conectan tangencialmente con los intereses y metas de los aprendices; la herramienta de análisis "Scrape" (Raspar), es de acceso público y la pueden usar los Scratchers para descubrir nuevos bloques; las entrevistas con frecuencia generan oportunidades de reflexión; para algunos aprendices, el diseño de escenarios representa rompecabezas intelectuales enganchadores y puede hacerse mucho más para que la evaluación tenga contexto y significado para el aprendiz. Este es un reto interesante pues lo que podemos valorar con mayor facilidad puede no ser lo más valioso para el aprendiz. Por ejemplo, un joven que está creando un juego interactivo quiere añadirle puntaje, pero no es consciente de los conceptos computacionales de variables y datos y, el tener a su disposición un listado de bloques de variables, puede no ser apoyo suficiente para alcanzar su objetivo.

SUGERENCIA #2: Incorporar artefactos

La evaluación debe incluir la creación y el examen crítico de proyectos. Los proyectos individuales son ejemplos ricos, concretos y contextualizados que pueden explorarse y analizarse en una diversidad de formas. Una colección de proyectos, enriquece mucho la valoración, pues ofrece la oportunidad de observar cómo la comprensión se desarrolla con el tiempo.

SUGERENCIA #3: Procesos reveladores

En nuestro primer enfoque de evaluación, utilizando el análisis basado en bloques, nos vimos limitados y nos dimos cuenta que las conversaciones productivas sobre los procesos de desarrollo van de la mano con los artefactos que se están creando. Enfocarse en el proceso ofrece una oportunidad para explorar el Pensamiento Computacional, que está representada de manera incompleta por los bloques: ¿Qué comprensión tiene el diseñador sobre conceptos particulares? ¿Qué prácticas emplea? La evaluación del proceso puede asumir múltiples formas y no requiere observación en tiempo real. Los jóvenes creadores computacionales pueden documentar sus procesos mediante comentarios en su código o en las notas del proyecto, hablar de sus experiencias en presentaciones, embeber descripciones de sus proyectos de Scratch en grabaciones de audio, guardar pantallazos de su proceso de desarrollo; enseñar a otros lo que saben o, comprometerse a dar una entrevista tanto en retrospectiva como en tiempo real. Cualquiera que sea la forma, las conversaciones acerca de su trabajo involucran a los jóvenes en actividades de metacognición, estimulándolos a reflexionar sobre su forma de pensar, capacidad esta importante para desarrollarse como aprendices auto regulados.

SUGERENCIA #4: Realizar controles en múltiples momentos

El Pensamiento Computacional no es un estado binario de estar presente o no estarlo en un punto determinado en el tiempo y cualquier enfoque para evaluarlo debe permitir describir de dónde viene un aprendiz, dónde se encuentra en el momento presente y dónde podrá estar en el futuro. Adoptar, para evaluar, un enfoque formativo, implica realizar controles en múltiples puntos durante la experiencia de aprendizaje computacional y puede involucrar también el realizar controles durante una actividad particular de diseño; tal como reunirse con un colega para hablar del progreso a medida que avanza la creación.

SUGERENCIA #5: Valorar múltiples formas de comprender

La intersección entre los conceptos y las prácticas del Pensamiento Computacional conduce a múltiples formas de saber. No es suficiente con estar en capacidad de definir un concepto, como ¿"Qué es un ciclo"? ¿Está realmente capacitado el aprendiz para usar ese concepto en el diseño de manera significativa? ¿Está capacitado el aprendiz para leer de qué manera otros han usado el concepto y luego remezclarlo para sus propios fines? ¿Está capacitado el aprendiz para analizar y criticar su propio código y el de otros? ¿Puede el aprendiz depurar un código que presenta problemas? La evaluación debe explorar múltiples formas de saber.

SUGERENCIA #6: Incluir múltiples puntos de vista

Nuestros segundos y terceros enfoques se enriquecieron considerablemente cuando la evaluación se deslindó del punto de vista único del investigador. En las entrevistas, por ejemplo, fueron posibles nuevos descubrimientos sobre el desarrollo de los Scratchers, que los padres o los pares pusieron en evidencia durante las conversaciones. En la comunidad en línea de Scratch, la retroalimentación y crítica de los pares se valora mucho. La evaluación debe acoger esta multiplicidad de puntos de vista, involucrando, como posibles y apropiadas, las evaluaciones de la persona, los pares, los padres, los maestros y los investigadores.

Estas sugerencias se basan en lo que analíticamente hemos detectado como los componentes más productivos de nuestros tres enfoques de evaluación y en los aprendizajes realizados en conversaciones tanto con jóvenes Scratchers como con educadores en Scratch. Esperamos que otros acepten estas sugerencias, así como los ejemplos de los tres enfoques y los remezclen para generar nuevas formas de evaluar.

RECURSOS ADICIONALES

- Scratch curriculum guide[6]
- Computational thinking concepts webinar[7]
- Computational thinking practices webinar[8]
- Computational thinking perspectives webinar[9]
- Assessing computational thinking webinar[10]

AGRADECIMIENTOS

Este material está basado en el trabajo apoyado por la National Science Foundation con la subvención No. 1019396. Las opiniones, resultados y conclusiones o recomendaciones expresadas en este material son las del autor (es) y no reflejan necesariamente los puntos de vista de la National Science Foundation.

REFERENCIAS

Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J., Martin, F. (2010). Computational thinking for youth. White Paper for the ITEST Small Working Group on Computational Thinking (CT).

Bandura, A. (2001). Social cognitive theory: An agentic perspective. Annual Review of Psychology, 52, 1-26.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? ACM Inroads, 2(1), 48- 54.

Brennan, K. (2011). Creative computing: A design-based introduction to computational thinking. Retrieved May 9, 2012, from http://scratched.media.mit.edu/sites/default/files/CurriculumGuide-v20110923.pdf

[6] http://scratched.media.mit.edu/resources/scratch-curriculum-guide-draft
[7] http://scratched.media.mit.edu/resources/computational-thinking-concepts-march-2011-webinar
[8] http://scratched.media.mit.edu/resources/computational-thinking-practices-april-2011-webinar
[9] http://scratched.media.mit.edu/resources/computational-thinking-perspectives-may-2011-webinar
[10] http://scratched.media.mit.edu/resources/assessing-computational-thinking-may-2012-scratched-webinar

Brennan, K., & Resnick, M. (in press). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In N. Lavigne & C. Mouza (Eds.), Emerging technologies for the classroom: A learning sciences perspective. Springer.

Brennan, K., Resnick, M., & Monroy-Hernandez, A. (2010). Making projects, making friends: Online community as catalyst for interactive media creation. New Directions for Youth Development, 2010(128), 75-83.

Brennan, K., Valverde, A., Prempeh, J., Roque, R. & Chung, M. (2011). More than code: The significance of social interactions in young people's development as interactive media creators. In T. Bastiaens & M. Ebner (Eds.), Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2011 (pp. 2147-2156). Chesapeake, VA: AACE.

Cuny, J., Snyder, L., & Wing, J.M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Kafai, Y. B., & Resnick, M. (Eds.). (1996). Constructionism in practice: Designing, thinking, and learning in a digital world. Hillsdale, NJ: Erlbaum.

National Academies of Science. (2010). Report of a workshop on the scope and nature of computational thinking. Washington DC: National Academies Press.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60-67.

Wolz, U., Hallberg, C., & Taylor, B. (March, 2011). Scrape: A tool for visualizing the code of Scratch programs. Poster presented at the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX.

CRÉDITOS:

Traducción realizada por Eduteka del documento "New frameworks for studying and assessing the development of computational thinking"[11] escrito por Karen Brennan (kbrennan@media.mit.edu) y Mitchel Resnick (mres@media.mit.edu), MIT Media Lab.
Versión HTML: http://www.eduteka.org/EvaluarPensamientoComputacional.php

Publicación de este documento en EDUTEKA: Septiembre 01 de 2012.
Última modificación de este documento: Septiembre 01 de 2012.

[11] http://scratched.media.mit.edu/resources/new-frameworks-studying-and-assessing-development-computational-thinking