UN LENGUAJE DE PROGRAMACIÓN PARA EL CUERPO ABORDAJE RETÓRICO SOBRE LA PROGRAMACIÓN Y SU ARTICULACIÓN CON LAS ARTES PERFORMÁTICAS.

Francisco Alvarez Lojo - Ezequiel Rivero - Daniel Loaiza -
Hernán González Moreno - Carolina Ojcius.
UNLP, FBA, Laboratorio EmmeLab, SeCyT

RESUMEN

El emmeLab es un Laboratorio de investigación y experimentación en nuevas interfaces para el arte, dependiente de la Secretaría de Ciencia y Técnica de la Facultad de Bellas Artes, UNLP. Durante el 2014 se planteó la exploración de generar un sistema programable que sea controlado a través de la captura de movimiento en tiempo real, en un intento de generar un entorno que permita una suerte de LiveCoding combinado con Danza Contemporánea o Expresión Corporal.
En primer lugar fue necesario generar un tipo de lenguaje de programación pensado específicamente para ser controlado mediante captura de movimiento. Realizamos una pequeña investigación respecto a los elementos que componen o definen un lenguaje de programación, planteamos algunos conceptos sobre las partes de un lenguaje y sobre paradigmas de programación reconocibles.
Finalmente decidimos crear un lenguaje que programe sistemas de partículas por diferentes razones. Por un lado nos pareció más facil que el resultado del lenguaje sea específicamente visual. Puede funcionar muy bien a nivel escenográfico y el emmeLab tiene mucha más experiencia en el trabajo generativo visual que en otros. También nos limitamos a un solo campo pues en experiencias pasadas la intención de abarcar demasiado causó una serie de complicaciones que distrajeron al proyecto de su objetivo principal. Por otro lado un sistema de partículas nos pareció muy adecuado porque su comportamiento está fuertemente ligado al movimiento. A su vez puede generar fácilmente una conexión semántica con la danza o expresión corporal donde el movimiento y el cuerpo son los protagonistas.
Respecto a la captura de movimiento utilizamos el sistema Kinect para tomar y pre-procesar la información. Definimos una serie de gestos o poses que serían detectadas como “entradas” para intervenir el “código”.
Tras una pequeña investigación en danza encontramos que existen ciertos parámetros “técnicos” en la disciplina que son utilizados para describir pasos y movimientos. Tratamos de explotar estos parámetros y al mismo tiempo intentamos que los gestos propios del lenguaje no estorbaran la posibilidad de una interpretación estética o natural por parte del performer.
Finalmente logramos generar un sencillo sistema de partículas muy maleable, aunque actualmente no ofrece mucha variedad. Sin embargo la estructura del sistema está diseñada para poder incrementar fácilmente y de manera colaborativa las posibilidades que ofrece. El sistema es controlado a través de mensajes OSC, estos mensajes pueden enviarse entre diferentes aplicaciones, de esta manera el control del sistema puede darse

por cualquier medio, más allá de que nuestra intención apunta a que sea utilizada con captura de movimiento.

PALABRAS CLAVE

Kinect, mocap, motion capture, interactivo, detección de movimiento

LIVE-CODING

El live-coding es una disciplina que consiste en transformar el acto de programar en un acto performático. El programador-performer se presenta ante un público y escribe con su herramienta predilecta un programa que genera visualización y/o musicalizaciones en tiempo real. La pagina web TOPLAP [1] es el principal referente de live-coding y en ella existe una suerte de manifiesto que fundamenta el live-coding. Allí se mencionan ciertas "buenas costumbres", como por ejemplo que es aceptable utilizar herramientas propias, pero un código debe generarse desde cero, y la fuente debe ser visible (incluso cuando el producto del programa es también visual). [1] [2] [3].
Aunque nuestra inspiración surge en parte del live-coding, no consideramos que se pueda clasificar este proyecto dentro de esa disciplina, pues buscamos generar una expansión de la danza hacia la tecnología, y no del live-coding hacia la danza. Nuestros criterios ya están bastante limitados por nuestra intención de no restringir al performer como para obligarnos a respetar otras limitaciones en un proyecto tan experimental.

LENGUAJES DE PROGRAMACIÓN

Antes de generar nuestro propio lenguaje hicimos una breve investigación para definir más claramente a que nos referimos cuando hablamos de un lenguaje de programación.
Primero, entendemos que un "programa" es un algoritmo; "un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución." [5]
Los algoritmos no son necesariamente para ser ejecutados por máquinas, las instrucciones dadas por un manual de cocina dirigidas a un ser humano también se consideran un algoritmo. De la misma manera que para poder transmitirle a alguien las instrucciones para alguna tarea es necesario hablar el mismo idioma, para poder dar instrucciones a una máquina u ordenador es necesario un lenguaje común. Esto sería el lenguaje de programación.
Normalmente, cuando hablamos de lenguajes de programación informático nos imaginamos una lista de palabras clave, símbolos y números escritos como texto sin formato, sin embargo no todos los lenguajes de programación se ven de esa manera. Lenguajes tales como Processing [6], PureData [7] y Piet [8] presentan claras deferencias en su representación.

En todos los lenguajes de programación encontramos dos operaciones indispensables para generar algoritmos, estructurar y parametrizar. Estructurar el algoritmo es definir las instrucciones que ejecutará el programa; el orden y la relación que hay entre ellas. Parametrizar es definir las variables que estas instrucciones requieren para tener un significado concreto. Para poder crear cualquier programa informático, necesitamos tener una herramienta (normalmente otro programa) que nos permita ejecutar estas operaciones en el lenguaje que estemos utilizando. Necesitamos poder ver y editar el algoritmo de alguna manera.

Teniendo esto en mente hicimos un análisis de las diferentes herramientas que permiten crear programas en diversos lenguajes, intentando tomar paradigmas variados para poder comprender que cosas eran comunes a todos ellos. Es pertinente aclarar que nuestro análisis no se concentra en las especificaciones técnicas de cada lenguaje -tratadas mejor por la Ingeniería Informática-, sino más bien una lectura humana de estos lenguajes prestando mayor atención a la interfaz y lo que la interfaz propone.

En todos los casos pudimos reconocer ciertos Elementos Generales que aparecían siempre, aun cuando los paradigmas de programación de cada lenguaje fueran diferentes. Estos elementos los definimos con los nombres de Materialidad, Representación, Primitivas, Parámetros, Estructuras y Espacios.

Con Materialidad nos referimos a la salida que el programa puede producir. Es común que existan herramientas que promueven ciertos tipos de salida sobre otras, pero no de manera exclusiva. Por ejemplo, tanto el lenguaje Processing como Pure Data pueden tener salidas en forma de video y/o audio, sin embargo el primero da mayores facilidades para ejecutar un resultado gráfico, mientras que el segundo provee facilidades para una salida sonora. Por otro lado, no dejan de existir lenguajes que estan asociados a un hardware específico, y por ende, su Materialidad es limitada por el mismo, como el caso de Arduino [9], que solo puede materializar sus programa a través de sus pines electronicos o su canal serie USB.

En contraste a la materialidad existe la Representación. Con esto nos referimos a la representación que se le da al algoritmo que estamos trabajando. Podemos arriesgarnos a decir que todos los lenguajes de programación tienen una representación visual, pero eso no impide que haya una gran variedad de representaciones, como vimos en las imágenes anteriores. En algunos casos la representación puede ser como una lista de instrucciones escritas linealmente, mientras que en otros casos podemos encontrarnos con un mapa de flujo que describe la estructura del algoritmo. Incluso podemos encontrarnos con el caso de Befunge [10] donde la representación del algoritmo es más bien como un laberinto formado por caracteres.

Las Primitivas son las instrucciones y operaciones elementales que ofrece el lenguaje, podríamos decir que son los ladrillos que tenemos a disposición para construir un algoritmo con él. Normalmente cuanto más sofisticadas son las primitivas de un lenguaje, más sencillo se vuelve generar programas complejos con éste. Al mismo tiempo, un lenguaje con primitivas básicas suele ser menos especializado.

Los Parámetros son la información que se provee a cada primitiva, en el caso que ésta nos lo permita, para que tenga una aplicación concreta. Normalmente las primitivas que existen en un lenguaje ofrecen cierta flexibilidad durante su ejecución, dándonos la posibilidad de ejecutarlos dentro de cierto rango de posibilidades. En cada ejecución los parámetros se combinan con las primitivas para definir concretamente una de esas posibilidades. Por

ejemplo, en Processing: consideremos la instrucción “ellipse” (indica dibujar una elipse), los parámetros son la posicion y el tamaño que tendrá la elipse dibujada.
Las Estructuras son las diferentes formas en que podemos relacionar las primitivas del programa y la propagación (lineal, reticular, recursiva, etc.) que tiene la ejecución de un algoritmo. Cada lenguaje de programación ofrece diversas formas de estructurar primitivas y cada programa construido con ese lenguaje es una estructuración específica. Las diferentes estructuras que ofrece un lenguaje limitan y condicionan la manera en que el flujo del algoritmo se desarrolla.
Los Espacios son la distribución que presentan los elementos del programa, los “espacios” en los que pueden estar. En un lenguaje de programación es normal que la representación y la materialidad ocupen espacios diferentes, pero existen casos donde se superponen, mostrando tanto la estructura del algoritmo y la materialidad de su resultado en una misma pantalla. Incluso, en algunos casos particulares como Recursive Drawing [11] la construcción del algoritmo y su mismo resultado son difícilmente diferenciables. Sin embargo con los “espacios” no nos referimos exclusivamente a la posición espacial. Por ejemplo, existen lenguajes de programación que tienen un “momento” de ejecución y otro de edición, mientras que otros lenguajes superponen el tiempo de edición y de ejecución. De la misma forma, en el segundo caso es muy común que la posibilidad de interactuar con el algoritmo en tiempo real sea a través de el mismo espacio de edición, pero en el primer caso esta interacción debe definirse previamente en el mismo algoritmo.

REACTABLE

Un caso interesante para tener en cuenta es el de reactable [12]. Es un lenguaje de programación que posee una forma híbrida entre virtual y física. Está diseñado para generar una materialidad musical o sonora, pero la representación del algoritmo es igual de interesante. Podríamos argumentar que en este caso la representación y la materialidad no son necesariamente aspectos separados.
Otro factor interesante es que las primitivas son objetos físicos, lo que implica una serie de efectos que no suelen existir en los casos puramente virtuales. Por ejemplo, el algoritmo tiene un límite muy concreto respecto a la cantidad de primitivas que puede utilizar, y más allá de eso, las primitivas ocupan un espacio físico, lo que significa que a medida que el algoritmo crece, se obstruye a si mismo.
Esta cualidad física también es aprovechada por reactable para trabajar otros elementos del lenguaje. Por ejemplo, los parámetros acaban siendo la posicion y rotación de cada primitiva. Otro ejemplo es que acompañando el espacio para el algoritmo activo, existe un espacio para lo inactivo. De esta manera el lenguaje utiliza los espacios como forma de indicar las primitivas que participan del algoritmo y las que no, literalmente, estando o no estando en él.

APROXIMACIÓN A CREAR UN LENGUAJE

Lo más difícil a la hora de empezar a definir un paradigma de lenguaje fue evitar caer en los lugares comunes. En los análisis que hicimos notamos que cuando un lenguaje estaba diseñado con una materialidad especializada se percibía el efecto en casi todos sus

elementos. Principalmente en las estructuras.También era importante tener en cuenta que nuestra intención no era generar un lenguaje de programación general, sino llevar a cabo un experimento que nos lleve a explorar un término medio entre Live-Coding y Danza. Por eso intentamos limitar fuertemente el alcance del lenguaje, aunque sea en esta primera instancia.
Elegimos crear un lenguaje orientado a generar sistemas de partículas que produzcan una representación visual estética. Nos limitamos a una materialidad visual porque nuestro grupo tiene más experiencia en ese área, lo cual nos permitió concentrarnos en el objetivo del proyecto.
Otro principio que definimos para realizar este lenguaje fue que exista la posibilidad de trabajar los algoritmos con diferentes niveles de detalle, pudiendo generar diferentes sistemas de partículas de una manera más clásica (estableciendo cada elemento básico manualmente) o de una forma más general (combinando algoritmos ya creados). En este segundo caso, no sería lo usual en los lenguajes de programación (que generan estructuras cada vez más grandes), sino fusionando dos sistemas de partículas con la lógica de los algoritmos genéticos produciendo un resultado no más grande, pero sí nuevo. Evitando de esta manera la complejización del algoritmo. Esto da al usuario/performer la posibilidad de elegir entre concentrarse más en el acto de generar un algoritmo, o de realizar una performance donde la realizacion del algoritmo sea algo secundario.
Un último factor que nos pareció importante fue diseñar el lenguaje de manera que un algoritmo nunca produzca errores, y cuando el programador/performer no especifique algún parámetro necesario debe existir una solución predefinida. Es decir, el sistema debe funcionar siempre, sin importar el cuidado con que se construya el algoritmo. Esto restringió el lenguaje aún más, pero no nos pareció un problema, pues nuestra intención no era generar una herramienta de uso general.

DISEÑANDO EL SISTEMA

Al principio tratamos de imaginar que tipos de primitivas y estructuras deberían ser necesarias para poder generar cualquier sistema de partículas de los más convencionales. Pero rápidamente nos encontramos con dificultades a la hora de definirnos concretamente, o de cumplir con los principios que nos habíamos planteado.
Acabamos tomando el camino inverso, idear cómo deberían ser los resultados del lenguaje y construir para poder llegar a una herramienta que permita generar ese tipo de resultados.
Una de las propuestas que más nos interesaba era que los diferentes algoritmos pudiesen fusionarse con la lógica de la combinación genética [13]. Esto quiere decir que el resultado de unir dos algoritmos no sea un algoritmo mayor que contenga a los originales, sino un algoritmo de complejidad similar, como resultado de conservar algunos de los elementos presentes en cada uno de los algoritmos originales.
A partir de esto imaginamos cuatro posibles sistemas de partículas relativamente variados, con sus comportamientos y representaciones. Luego, analizamos que resultados deberían producirse -o ser posible que se produzcan- al fusionar dos de ellos de la manera que planteamos previamente.

De esta manera nos fue mucho más fácil entender cuáles serían las primitivas de cada algoritmo, cuáles elementos serían divisibles y cuáles no. También nos permitió entender cómo podríamos categorizar estos elementos.

EL SISTEMA DE PARTÍCULAS

Como la mayoría de los sistemas de partículas existentes, nuestro sistema de partículas tiene una cantidad constante de partículas las cuales comparten un mismo comportamienta. En cambio, a diferencia de la mayoría, la cantidad y tipo de atributos que tienen las partículas es dinámico, adaptándose a medida que generamos su “programa”.
Una cantidad constante de partículas significa que no se crearán o destruirán partículas durante la simulación, simplemente se define la cantidad inicial y esa cantidad se mantiene constante. Hay que tener en cuenta que a las partículas puede aplicárseles un comportamiento que “active y desactive” algunas de ellas, generando la ilusión de una cantidad dinámica de partículas.
Las partículas de un sistema son procesadas todas juntas por un solo programa, es por esto que tienen todas un mismo comportamiento. Sin embargo varios sistemas de partículas pueden convivir, generando variedad de comportamientos. Además, el sistema de partículas creado durante este proyecto fue diseñado para ofrecer la posibilidad de interacción entre varios sistemas de partículas, y para soportar condicionales. De esta manera el programa puede estar bifurcado internamente para generar comportamientos variados con un mismo sistema. Sin embargo, estas propiedades no han sido implementadas aún.
Los atributos fueron diseñados como dinámicos para crear un sistema que sea modular pero además escalable. Es decir, puede crearse un comportamiento que utilice un atributo que no exista en el sistema actual. De esta manera se abre la posibilidad a un desarrollo permanente y colaborativo de las posibilidades que ofrece el sistema. Hay que tener en cuenta que esto requeriría la creación de algún sistema que administre estas extensiones fácilmente.

GUÍA RETÓRICA

Cuando empezamos a buscar la “forma” del lenguaje, nos detuvimos para crear una suerte de metáfora guía. Previamente, en el momento que analizamos otros lenguajes, notamos que algunos de ellos parecían tener ciertas lógicas simbólicas que atraviesan todos o la mayoría de los elementos del lenguaje. Por ejemplo, en Pure Data podemos ver una clara referencia a la lógica de los procesadores de audio analógicos, basados en filtros que se conectan a modo de árbol, o cascada a la señal de origen. Otro caso es Scratch [14] que hace referencia a los juguetes de construcción con bloques o rompecabezas, donde cada bloque tiene una suerte de bordes que por su figura implican donde es posible ubicarlos.
En nuestro caso comenzamos buscando una retórica que tenga alguna asociación con el movimiento, en un intento de generar cierta coherencia con el hecho que nuestro lenguaje se orienta a un uso a través de la danza.
Tras algunas propuestas, elegimos una retórica inspirada en los proyectores fílmicos o grabadoras analógica, donde una cinta que contiene información circula continuamente a través de mecanismos que hacen algo con ella. A partir de esto empezamos a darle una forma un poco más concreta a las primitivas y lógicas propias del sistema.

PARADIGMA GENERAL

El lenguaje se basa en dos elementos claves: una cinta o canal de información, y un sistema de mecanismos modificadores. Estos mecanismos tienen efectos relativamente concretos, relevantes, pero pueden abrirse para revelar poleas, componentes más elementales del sistema que por sí solos no generan un efecto importante. En otras palabras, las poleas se combinan para formar mecanismos. Algunos de estos mecanismos tienen el efecto secundario de producir una imagen, en cuyo caso los apodamos proyectores. Otros mecanismos funcionan enmascarando parcialmente la cinta dando la posibilidad de afectar grupos concretos de partículas

IMPLEMENTACIÓN

Como lenguaje de base utilizamos Processing y lo primero a lo que nos abocamos fue a tener un sistema de partículas que cumpla con nuestros requerimientos aunque sea mínimamente. Luego generamos una sencilla API, es decir, una forma abierta del sistema de intercambiar mensajes con otros programas. Y finalmente creamos un programa que utiliza la Kinect [15] e interpreta ciertos parámetros en la pose del usuario para usar como elemento de entrada al lenguaje. La ultima versión del codigo esta alojada en *https://github.com/emmelab2014/Nucleo*.
El estado del sistema no alcanza la mayoría de las características finales que buscamos como la posibilidad de fusionar dos mecanismos, la capacidad de abrir un mecanismo y modificar sus elementos básicos o la capacidad de afectar grupos específicos de partículas, pero sí cumple con los fundamentos: El sistema puede fácilmente agregar, quitar o deshabilitar mecanismos, y los atributos de las partículas son generados dinámicamente en base a las necesidades de éstos mecanismos.
Sin embargo esta prueba inicial aportó una gran cantidad de información; por un lado demostró la viabilidad del lenguaje, aparecieron los primeros problemas técnicos concretos de su realización lo cual nos permite la posibilidad de reconsiderar ciertas cuestiones del primer diseño sugerido en base a datos concretos.

OBJETOS

Existen tres tipos de objetos clave en el programa actual: Sistema, Atributo y Modificador.
Sistema es el objeto que define un sistema de partículas específico. Contiene un ArrayList de Modificadores definido como “modificadores” y un HashMap de Atributos definido como “atributos”. Un ArrayList es simplemente una lista, los modificadores pueden repetirse en esta lista, y son ejecutados en el orden que posee. Podríamos decir que es el recorrido que hace la cinta, o que es el programa del sistema de partículas. Un HashMap es una suerte de diccionario, una lista de objetos asociados a palabras claves, sin orden concreto y donde una misma palabra clave no puede aparecer dos veces. Podríamos decir que este HashMap es en sí mismo la cinta.

Atributo es un objeto que sirve como esqueleto para la creación de los posibles atributos que pueden llegar a tener las partículas de un sistema, y son la información que un Modificador afecta. Cada Atributo es una extensión de este objeto padre, y todos poseen un String clave, un nombre que necesariamente debe ser único entre todos los posibles Atributos que existen en el lenguaje. Todos los Atributos deben tener un valor inicial por defecto.
Modificador es otro objeto esquemático que es extendido para generar los diferentes modificadores o procesos del programa. Actualmente el modificador está construido y tiene el comportamiento que deberían tener las poleas según nuestro diseño inicial, pero el efecto que tienen es más parecido al que teníamos planeado para los mecanismos. Cada Modificador define aquellos Atributos que requiere necesariamente para funcionar, y aquellos que son opcionales.

1. http://toplap.org/
2. http://networkmusicfestival.org/nmf2012/programme/performances/silicone-bake/
3. http://rtigger.com/blog/2013/05/15/live-coding-good-or-bad
4. http://www.lighthouse.org.uk/programme/brighton-digital-festival-closing-performances
5. http://es.wikipedia.org/wiki/Algoritmo
6. https://processing.org/
7. http://puredata.info/
8. http://www.dangermouse.net/esoteric/piet.html
9. http://www.arduino.cc/
10. http://es.wikipedia.org/wiki/Befunge
11. http://recursivedrawing.com/
12. http://www.reactable.com/
13. http://natureofcode.com/book/chapter-9-the-evolution-of-code/
14. https://scratch.mit.edu/
15. http://es.wikipedia.org/wiki/Kinect