Autoorganización y Evolución Biológica en la Música: Anexos

Santiago Diazgranados Berenguer

Universidad El Bosque
Estudios Musicales
Bogotá, Colombia, 2014

Índice de Contenidos

1. Registro de potenciales simuladores evolutivos para el proyecto

Las siguientes son reseñas breves de algunos de los muchísimos programas de vida artificial que existen actualmente. Provienen de la evaluación que se ha hecho sobre los mismos para su potencial utilización en el producto artístico, no mediante su utilización sino mediante a las descripciones de ellos.

Darwinbots

Simulación evolutiva en la que organismos virtuales denominados *bots* coexisten en un mundo virtual compitiendo unos con otros por comida.

- Funciona con algoritmos de ADN ejecutable.
- Los organismos evolucionan pero son asexuales.
- En el ADN de los *bots* está codificado su comportamiento. Este comportamiento los lleva a buscar energía para suplir sus necesidades metabólicas.
- Los *bots* mueren si tienen muy poca energía o demasiados desechos en su entorno. La eficiencia metabólica y el manejo de los desechos es lo que dirige el proceso evolutivo.
- Los *bots* son territoriales.
- Cada *bot* ejecuta todas las instrucciones de su genotipo en cada ciclo.
- Los *bots* coexisten en un espacio bidimensional que no es una cuadrícula como en el caso de *Conway's Game of Life*, sino un continuo análogo a una superficie lisa real.
- Los *food bots* (comida) son como las plantas: el primer piso trófico.

Contras

Los *bots* son muchos y se mueven muy rápido. Surge la incógnita de cómo va a representarse el movimiento de los bots en la posición (su cercanía con respecto al foco de perspectiva podría representarse por la intensidad del sonido como en *Seaquence*).
Es difícil percibir el cambio evolutivo entre las generaciones de organismos. Este podría ser más fácil de musicalizar que las relaciones interespecíficas entre los bots. Si bien sería interesante musicalizar esas relaciones de alguna manera, es más viable que estas sean determinantes para las características sonoras de los organismos que van surgiendo; y que provengan de las características de los organismos en sí.

wiki.darwinbots.com/w/Main_Page

3D Virtual Creature Evolution

Simulación evolutiva creada por Lee Graham.

- Procesos determinados mediante algoritmos Neural Net.
- Se crean criaturas tridimensionales en base a un esquema corporal basado en prismas rectangulares.
- El usuario determina cosas como el máximo número de tipos de segmento y el tamaño de los segmentos.
- Hay reproducción sexual y asexual. Los cambios se dan tanto por la mezcla de los genomas de los padres como por mutaciones.

- Se evalúa el *fitness* (calidad genética) de las criaturas en base a ciertos parámetros concretos (distancia que recorre, altura que salta, etc.).

- Hay distintos tipos de terreno (plano, montañoso, líquido).

- Las criaturas con el más alto valor de *fitness* se reproducen y pasan sus genes a la siguiente generación. Este valor va de 0 a 1.

- El usuario puede alterar la función del *fitness* para simular así eventos ambientales que afectan a las criaturas.

- Se pueden duplicar linajes para simular el fenómeno de especiación.

Contras

Ocurre lo opuesto que en *Darwinbots*. Se centra demasiado en los individuos y no lo suficiente en las relaciones que ocurren entre ellos.

http://3dvce.wikia.com/wiki/Main_Page

Blind Watchmaker

Software evolutivo simple diseñado por el zoólogo Richard Dawkins.

- Un set de genes se manifiesta en figuritas llamadas *biomorfos.* Este programa muestra con claridad como mutaciones aleatorias seleccionadas de forma no aleatoria generan patrones complejos afectados o no por una vertiente teleológica.

- En una cuadrícula aparece una serie de biomorfos. El biomorfo de la esquina superior izquierda es el biomorfo padre. Sus hijos están en las otras celdas de la cuadrícula, y cada uno es un poco distinto a su padre y a sus hermanos. El usuario puede seleccionar cuál de los hijos será el biomorfo padre en la siguiente generación. Al repetir este proceso varias veces, se empieza a ver un cambio gradual en la estructura de los

biomorfos que van volviéndose en ocasiones más complejos e interesantes. En la esquina inferior derecha hay una cuadrícula dentro de una de las celdas de la cuadrícula más grande. Acá se puede alterar la longitud y la dirección de cada gen sobre el biomorfo padre, así como el número de segmentos o niveles nuevos en el proceso de ramificación distintivo de los patrones fractales que el programa utiliza para el diseño evolutivo de los biomorfos.

Pros

Los biomorfos son agradables a la vista. Los genes son fácilmente codificables.

Contras

Tal vez es demasiado simple. No hay interacción competitiva entre los individuos más que la que ocurre en la mente del usuario que lleva a cabo el proceso de selección.

www.annanardella.it/biomorph.html

Creatures

Juego diseñado por Steve Grand para *Millenium Interactive.*

Fig. 1 Interfaz del juego (http://www.dotemu.com/sites/default/files/product/screenshots/creatures-exodus-pc-windows-screenshots__415_0.jpg)

- Comportamiento de los agentes determinado por algoritmos de redes neurales.

- Simulación biológica y neurológica que arroja resultados inesperados.

- Puede haber reproducción sexual.

- Las criaturas *(norms)* poseen apetitos que pueden ser saciados.

- Pueden ocurrir mutaciones en los genomas de los *norms*.

- Los *norms* aprenden y reaccionan a los estímulos en forma inteligente.

- El comportamiento complejo en las criaturas surge de principios simples en forma emergente.

Contras

Demasiado lúdico. Es básicamente un juego para niños.

DigiHive

Medio ambiente cerrado que contiene estructuras individuales con complejidad interna que demuestra comportamientos evolutivos.

- Funciona con algoritmos de ADN ejecutable.
- Hay unas entidades hexagonales que denominadas partículas. Estas habitan un espacio bidimensional continuo.
- Las partículas se mueven y colisionan entre sí, quedándose pegadas a veces para conformar estructuras complejas.
- Las estructuras se reconocen unas a otras e interactúan entre sí.
- Utiliza un lenguaje de programación que se llama *Prolog*.

Contras

Las estructuras carecen de ejes axiales y líneas de simetría que podrían delimitar la apropiada conversión de ellos a componentes sonoros.

digihive.pl

Techno Sphere

Medio ambiente digital en línea creado por Jane Prophet y Gordon Selley.

Fig. 2 (http://en.wikipedia.org/wiki/TechnoSphere#mediaviewer/File:Technosphere4.jpg)

- Funciona por medio de algoritmos de módulo.

- En un ecosistema virtual en línea, usuarios de todo el mundo crean criaturas y las dejan libres para que interactúen unas con otras. Está basado en algoritmos de teoría del caos.

- Hay surgimientos de patrones inesperados. Por ejemplo, aunque no hay un algoritmo para que las criaturas formen manadas, a veces lo hacen.

- La topología del terreno está creada por un sistema de paisajes fractales.

Contras

Es un juego multiusuario. Las criaturas se crean ya complejas en lugar de evolucionar desde formas más simples.

http://web.archive.org/web/20030625071905/http://www.technosphere.org.uk/

Breve

Simulador de vida artificial donde el usuario puede definir los comportamientos de los individuos para que se manifiesten en un medio ambiente tridimensional, posibilitando además la observación y el registro de los comportamientos de los agentes cuando interactúan entre sí.

Utiliza un lenguaje de programación denominado *steve* que se caracteriza entre otras cosas por ser fácil de usar. El programa se puede descargar de forma gratuita en la página oficial.

Hay *plug-ins* en *Breve* para generar música por medio de la interfaz MIDI.

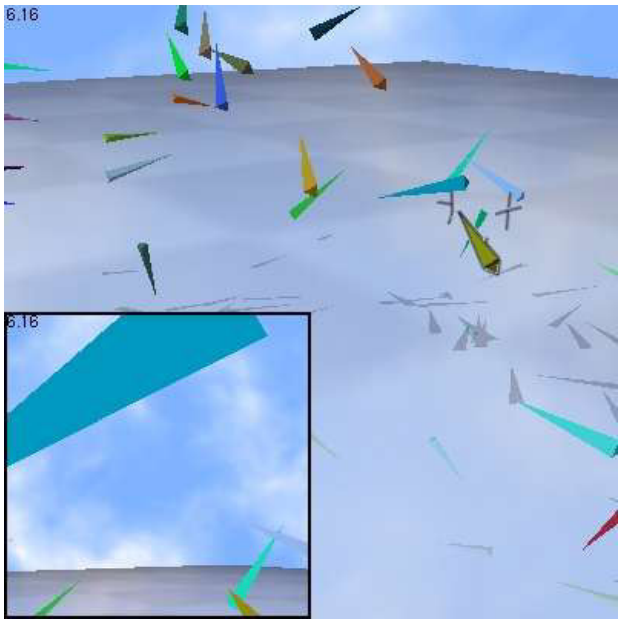


Fig. 3 Enjambre de agentes hechos en *breve* que se agrupan mediante algoritmos que determinan este tipo específico de comportamiento. En la esquina inferior izquierda vemos la perspectiva de uno de los agentes en particular (http://www.spiderland.org/breve)

Contras

Se hacen pocas referencias en las descripciones disponibles al proceso evolutivo en el ecosistema, es decir, a la evolución en términos de relaciones ecológicas.

Noble Ape

Simulación de un hábitat biológico. En este viven agentes denominados *noble apes* (simios nobles). Tiene la peculiaridad de que los *apes* llevan a cabo procesos cognitivos, por lo que es también un programa que trabaja con algoritmos de inteligencia artificial.

Los *noble apes* se mueven y reaccionan a eventos externos como el clima y la presencia de otros *apes*, así como a fenómenos intrapersonales como el miedo, el pánico y el sueño.

Los ecosistemas geográficamente realistas y muy complejos.

Contras

No hay indicios de que ocurra una evolución gradual a partir de entidades simples. Más bien parecería que los organismos emergen ya en un estado de complejidad avanzada.

http://www.nobleape.com/

Tierra

Simulación por computador donde distintos programas compiten por tiempo y acceso al procesador central de un computador (*CPU*). Mediante presiones asociadas a esta búsqueda por parte de los programas ocurren fenómenos análogos a los observados en la evolución biológica tales como mutación, reproducción y recombinación de características.

Al no haber un sistema en el que se vuelve más probable la supervivencia y consecuente reproducción entre los individuos con un nivel más elevado de *fitness*, no hay realmente un proceso de optimización genética por lo que ocurre entonces una evolución de cabos sueltos (una evolución donde las características de los agentes involucrados emergen en forma aleatoria y no en virtud de resultar adaptativas). Es por esto que a veces es demasiado reiterativa y cíclica, y es en esto que se distingue de la evolución biológica observada en el mundo real.

http://life.ou.edu/tierra/

Framsticks

Framsticks presenta un hábitat tridimensional donde habitan agentes construidos con varas unidas por ligamentos con músculos que mediante una red neural determinan el comportamiento del agente. Estos se alimentan de orbes de energía que están dispuestos a lo largo y ancho del mapa de forma deliberada por el usuario. Los agentes tienen cerebros y operan mediante algoritmos simples de inteligencia artificial.

El proceso evolutivo es dirigido por el usuario, es decir, funciona mediante procedimientos de selección artificial. No obstante, el usuario no escoge a unos agentes sobre otros según su preferencia como en *The Blind Watchmaker* sino que predetermina mapas de *fitness* para que sea seleccionada una aptitud en particular como la velocidad del agente, por ejemplo.

Los genomas de los agentes pueden ser alterados cualquier momento por el usuario.

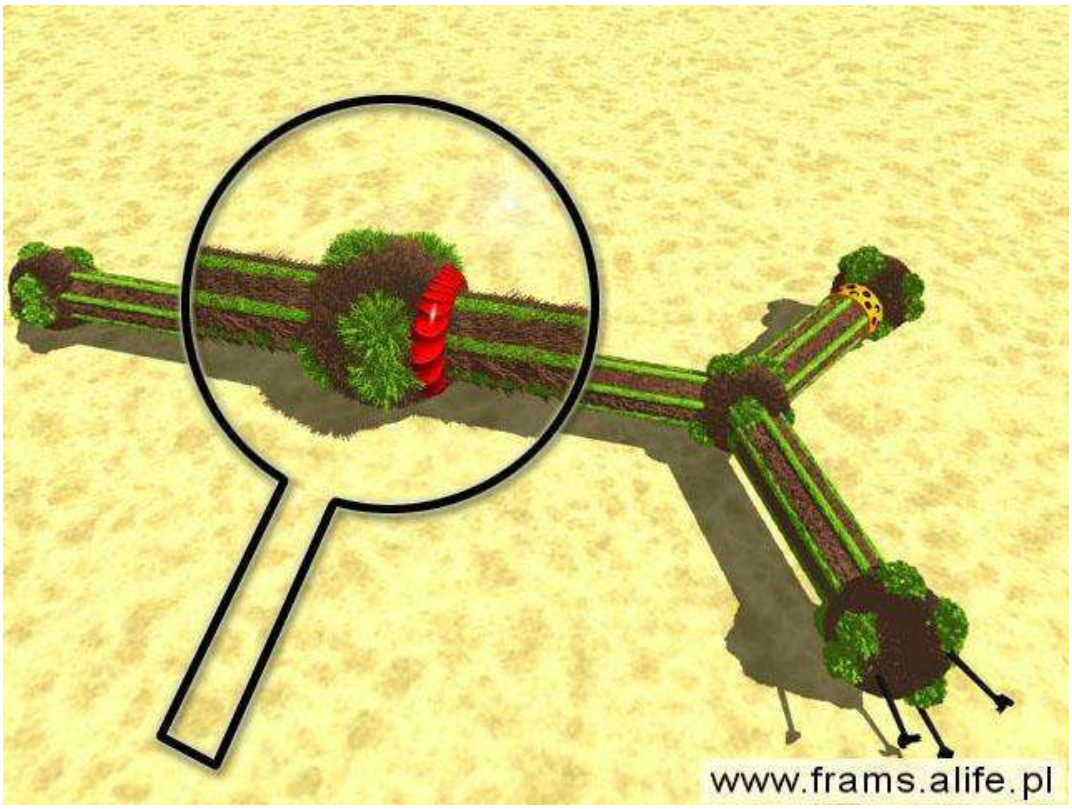


Fig. 4 Organismo segmentado con un músculo y receptores sensoriales en forma de antenas. Este organismo en particular no evolucionó sino que fue creado para la demostración de la apariencia de los organismos en el programa. (http://www.framsticks.com/)

Contras

El proceso evolutivo ocurre por selección artificial y no por selección natural.

http://www.framsticks.com/

Eco Sim

Ecosistema virtual basado en modelos naturales de depredación. El proceso evolutivo que en él ocurre determina el comportamiento de los agentes a lo largo del mismo. Hay un mecanismo de especiación que puede ser acelerado por la presencia de obstáculos tipo barrera geográfica.

El genoma de los agentes está representado por un *Mapa Cognitivo Borroso* (*Fuzzy Cognitive Map*). Este determina el comportamiento del agente con respecto a circunstancias como cercanía de este a una fuente de alimento, cercanía de este a un depredador, estados internos como hambre y miedo, y conceptos motores como escape y reproducción.

Estos factores se afectan unos a otros generando cadenas y redes complejas de actividad cognitiva. Por ejemplo, un depredador en las cercanías genera miedo en sus presas por lo que desencadena en ellas el uso de las aptitudes de escape.

La información genética se trasmite de generación a generación mediante la herencia y se recombinan mediante reproducción sexual. A lo largo del proceso evolutivo surgen nuevos significados para estos conceptos genéticos (las aletas de los pescados que

pasaron de ser apéndices para desplazarse en el agua a ser apéndices para desplazarse en la tierra son un ejemplo de esto en el mundo natural).

El programa permite la realización de un registro de los eventos de especiación a lo largo del tiempo en forma de árbol filogenético.

Cada agente posee características como: edad máxima, edad mínima para aparearse, tamaño del campo de visión, niveles máximos de energía y cantidad de energía trasmitida a su descendencia.

El alimento de los agentes proviene del pasto (primer piso trófico) o de carne (proveniente de otros agentes). Las acciones de los agentes necesitan energía para ser llevadas a cabo, y esta energía proviene del alimento que consumen. Si un agente agota toda su energía, este muere. La manera en que están diseñadas las cadenas de alimentación posibilita la aparición de relaciones simbióticas. Una "jugada" (*run*) del programa puede durar miles de generaciones.

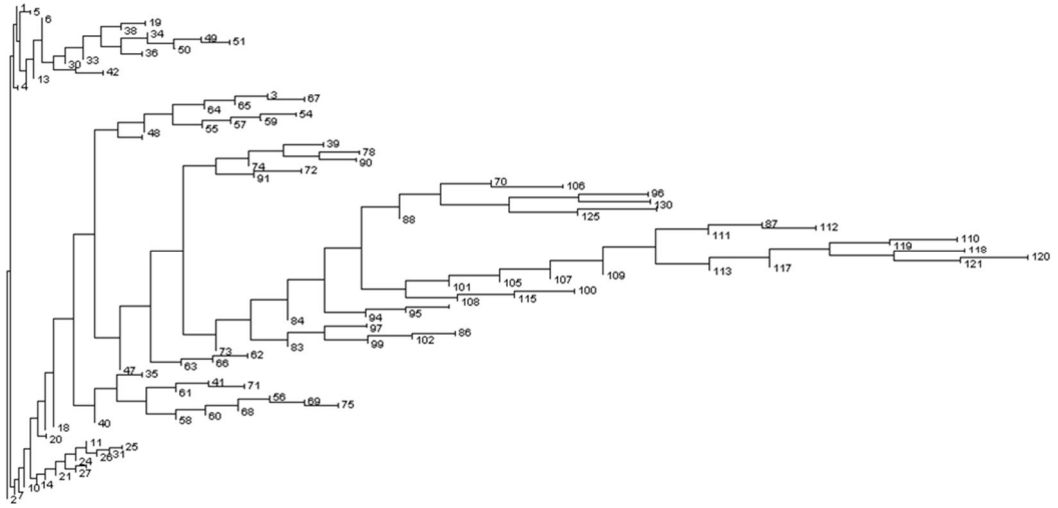


Fig 5. Árbol filogenético que surgió de una jugada (*run*) de *Eco Sim*.

Pros

-Involucra depredación.

-Hay especiación (aparición de nuevas especies).

-Hay obstáculos físicos que ayudan a que se dé una simulación de los fenómenos asociados al aislamiento geográfico como una mayor probabilidad de especiación.

-Ocurren relaciones no lineares en diferentes niveles del ecosistema (cognición, relaciones interespecíficas) que pueden sugerir la presencia de fenómenos emergentes aptos para la comprensión sistémica.

-De todos los programas vistos hasta ahora este permitiría llevar a cabo la mayor cantidad de consideraciones en base a las intenciones específicas que se tienen para el producto artístico.

Contras

-Podría ser en realidad demasiado complejo.

-Podría funcionar con magnitudes tan grandes que resultarían poco adecuadas para ser trabajadas de manera musical.

https://sites.google.com/site/ecosimgroup/research/ecosystem-simulation

2. Bitácora del Proyecto Artístico

En esta sección se ha elaborado un registro del proceso de desarrollo del producto artístico, es decir, de la composición musical. Se describen acá los pasos de este proceso, incluyendo aquellas soluciones que se exploraron pero no se implementaron finalmente. Los experimentos que dieron como resultado estos sistemas y esquemas que finalmente no se implementaron fueron parte fundamental de proceso. Lejos de ser equivocaciones en la metodología, en su consideración se documenta la presencia de los eslabones del proceso evolutivo del sistema. Esto es particularmente claro cuando se observan las imágenes de estas etapas que se fueron desechando para dar paso a formas más eficientes de realizar los procesos que se acomodarían a la intención del compositor.

En proceso evolutivo se mantuvieron ciertas características de los sistemas para ser implementadas en los sistemas que las seguían; y se descartaron otras que constituyeron callejones sin salida evolutivos. Dado que se aprendió a utilizar el programa Max solo para ser utilizado en este proyecto, la construcción del *patcher* fue un proceso de tanteo miope. En los eslabones, es decir, en las distintas generaciones en la evolución del *patcher*, quedó registrada esta "torpeza exploratoria", así como en las formas que toma la vida biológica queda registrado el hecho de que la evolución biológica es un proceso ciego; un proceso de *diseño no-inteligente* (en oposición al Diseño Inteligente de los Creacionistas). Con frecuencia, los sistemas tomaron formas estrambóticas que a veces constituyeron procesos redundantes que serían corregidos en virtud de la función de *fitness* que es la intención del compositor: crear una composición generativa-evolutiva.

Esto no es deliberado, es decir, no constituye una actitud intencional a resaltar la cualidad evolutiva, dado que el tema de la composición y el escrito son la evolución. Es más bien una reflexión *a posteriori* que bien podría aplicar al diseño de cualquier sistema, objeto, u construcción literaria.

También hay aquí un registro de las exploraciones didácticas, es decir, de los procesos de aprendizaje que se llevaron a cabo exclusivamente para este proyecto. Ejemplo de esto son el registro del aprendizaje del lenguaje Java y de la exploración de programas como *Darwinpond* y *NetLogo*, los cuales finalmente no se implementaron.

8 de abril 2013

De las opciones sobre las que se ha obtenido información parece ser *EcoSim* la más apropiada, ya que maneja el mayor número de variables asociadas a la evolución en ecosistemas reales. Muchas de estas variables las tomé en cuenta cuando elaboré la descripción del producto artístico. Adicionalmente parece ser el más moderno entre los programas de este tipo. Lo que destaca a *EcoSim* por sobre otros programas de vida artificial es que trabaja con un sistema parecido a una red neural donde interactúan diferentes factores en los individuos como deseos, miedos, acciones y percepción. Este sistema se llama *Fuzzy Cognitive Map* (FCM). El *FCM* de cada individuo del ecosistema es único y es un producto del desarrollo evolutivo. Según los creadores del programa, *EcoSim* es la única plataforma que combina los factores de especiación con los del comportamiento individual de los agentes.

EcoSim produce datos en tres niveles distintos. El primero de los tres es el de enfoque más amplio. En él los datos corresponden a estadísticas globales como el número de individuos, el número de especies y la cantidad de alimento disponible. Al segundo nivel le conciernen estadísticas al nivel de la especie tales como el *FCM* general de la especie, el nivel promedio de fitness y el número de individuos en esa especie en particular. El tercer nivel se enfoca en cada agente de forma individual y le conciernen datos con respecto al nivel de energía y la edad del agente entre otros.

Se procede a descargar un paquete que incluye *Ecopath* (una imagen estática del ecosistema), *EcoSim* (el simulador del ecosistema en sí) y *Ecospace* (un módulo espacio temporal diseñado para la exploración de impactos ecológicos.) Se hará la descarga desde www.ecopath.org.

9 de abril 2013

El programa no era el que se necesitaba. Se llamaba igual pero no era el mismo. Se descargó otro que también llevaba el mismo nombre pero tampoco era el simulador evolutivo como tal. Ambos estaban relacionados con ecosistemas pero no en la forma en que me serviría. Se ha escrito un email al inventor de *EcoSim*, Robin Gras en el que se le explican las intenciones del proyecto y se le pregunta sobre las posibilidades de adquisición del programa.

Hay un plan B: un programa que se llama *Polyworld*. Este utiliza una red neural más simple que la de EcoSim por lo que el comportamiento de los agentes es menos complejo (aunque lo sigue siendo no obstante). Tampoco hay mención en las descripciones de este programa de ocurrencia de eventos de especiación a lo largo del transcurso evolutivo simulado.

13 de abril 2013

Se ha encontrado un programa que se llama *Maestro Genesis*. Este programa utiliza un algoritmo evolutivo básico para ayudarles a músicos principiantes a generar acompañamientos para canciones MIDI. Es parecido a *The Blind Watchmaker* en el sentido que refleja los mecanismos de la selección artificial. En *The Blind Watchmaker* uno escoge un *biomorfo* para que sea el padre de la siguiente generación de biomorfos, los cuales tendrán características suyas al mismo tiempo que implementan variaciones ligeras. *Maestro Genesis* le permite al usuario escoger entre una serie de acompañamientos para que aquellos que sean escogidos determinen las características de la generación siguiente. Se pueden escoger dos y combinarse, en un caso reminiscente a la reproducción sexual. A diferencia de la reproducción sexual, se pueden escoger más de dos acompañamientos, guardando entonces todas las opciones de la siguiente generación características de los acompañamientos que fueron escogidos en la anterior como en una suerte de reproducción sexual con más de dos individuos involucrados. Cuando después de cierto número de generaciones el usuario está contento con el acompañamiento actual, puede guardarlo en forma de mp3. Se pueden guardar varios acompañamientos en cualquier punto del proceso para que más tarde el usuario escoja el mejor.

Tanto en *The Blind Watchmaker* como en *Maestro Genesis* ocurre un proceso de selección artificial como el que implementan los criadores de animales domesticados. Un criador de perros hará aparear a los perros que reflejen mayores aptitudes para un determinado propósito para que su descendencia conserve dichas características. Es

mediante este proceso que se han criado razas de perros tan distintos como el chihuahua y el gran danés a partir de lobos grises. Este proceso está reflejado en *Maestro Genesis* permitiendo que el usuario elija los acompañamientos que más se acoplan a sus necesidades para que pasen características a su descendencia hasta que esta sea óptima para los propósitos del usuario.

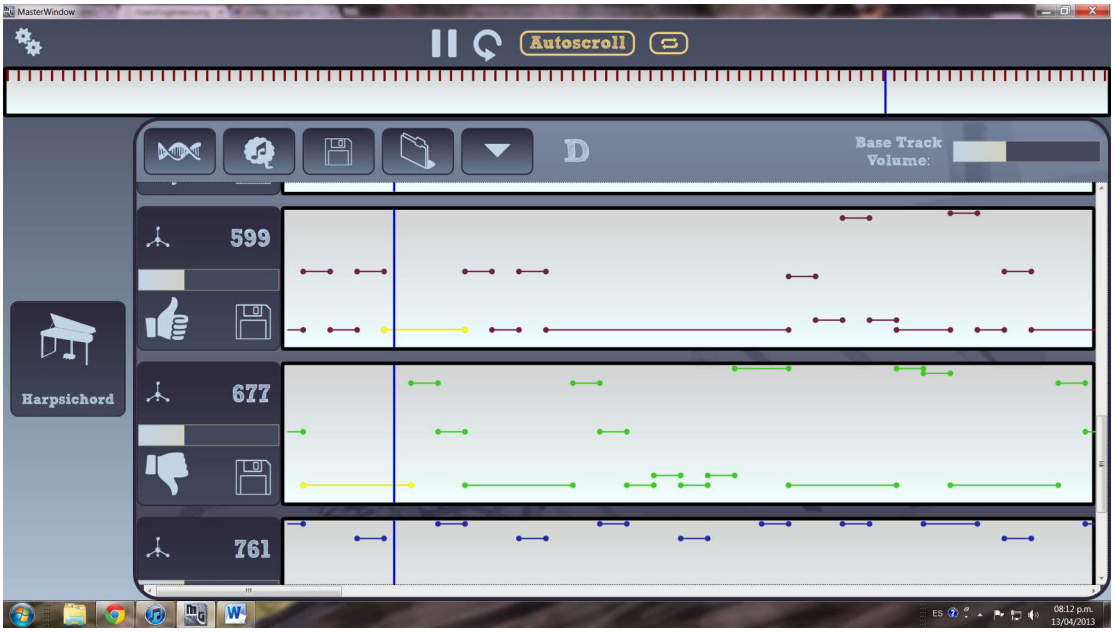


Fig. 6 Interfaz tipo secuenciador de *Maestro Genesis* (Maestro Genesis, 2012).

Hay muchas razones por las que este programa no funcionaría para la realización del actual producto artístico. Una es que (al igual que en *The Blind Watchmaker*) utiliza un sistema de descendencia que requiere selección artificial, mientras que uno de los requerimientos de la composición es que la evolución se debe dar a través de la selección natural, es decir, sin intervención del usuario o compositor.

23 de abril 2013

Robin Gras, creador de *EcoSim* respondió el email que le fue enviado en virtud de la presente investigación. Dice que *EcoSim* no se puede descargar y los sets de datos que arroja son demasiado pesados para un computador personal. No obstante ofreció mandar algunos de estos datos si son requeridos. Se tomará en cuenta este ofrecimiento en caso dado de que sea útil pero se optará primordialmente por la exploración de otros programas.

Adicionalmente se descargará un programa llamado *Species*. Es un programa moderno y con gráficas relativamente buenas que imita fenómenos evolutivos. El programa está en sus etapas iniciales lo que es bueno pues se puede descargar de manera gratuita.

No se debe olvidar que hay muchas otras opciones incluidas las expuestas en el anteproyecto. Hay una que no se incluye entre estas y se llama *Darwinpond*. Su descripción suena interesante y sería bueno tenerla en cuenta.

24 de abril 2013

Se descargaron dos versiones distintas de *Species*. Ninguna de las dos funcionó. Entre los links del video de demostración de *Species* en YouTube hay un link para la demostración de un programa que se llama *Darwin Pond*. Se vieron los videos (6 en total) y se llegó a la conclusión de que *Darwin Pond* es un buen candidato para la elaboración del proyecto. Se descargó la última versión del programa en www.verdella.com. El programa simula un charco de agua donde conviven criaturas denominadas *swimmers*. Estas compiten por alimento y se reproducen evolucionando

configuraciones morfológicas y de comportamiento que son pasadas a través de las generaciones.

Al mismo tiempo que el programa está siendo explorado se está leyendo el manual para sacársele el mejor provecho posible a la experiencia. Se han visto muestras de pantanos (*ponds*) con características iniciales predeterminadas como la muestra *Race* en la que dos poblaciones distintas siguen senderos de comida que llevan a un depósito de alimento más grande donde interactúan, compiten o se aparean.

Los *swimmers* van perdiendo intensidad en la coloración de sus cuerpos a medida que van perdiendo energía. También se mueven más despacio a medida que esto ocurre. Un *swimmer* famélico con un nivel de energía de 1, está al borde de la muerte, pero el usuario puede salvarle la vida si lo arrastra con el mouse hasta una fuente de alimento.

28 de abril 2013

Se habló con la bióloga graduada de los Andes, Lina María Valencia, quién trabajó en su proyecto de grado con un programa que permite la simulación de ecosistemas virtuales. El programa se llama *NetLogo*. Está basado en agentes dinámicos. Cada agente tiene un set de variables o características y responde al ambiente y a los demás agentes en el mismo ambiente. Cada agente basa su comportamiento también en una serie de reglas jerárquicas asignadas por el usuario. El programa muestra diversos niveles de complejidad en los agentes y en el hábitat virtual. En un estado avanzado de un sistema, los agentes pueden tener incluso memoria y procesos simples de aprendizaje vinculados a su comportamiento.

Recomendó la utilización de los foros del programa pues acá se han dado respuestas a muchas inquietudes comunes que tienen los usuarios sobre el programa. Este es el link del programa: http://ccl.northwestern.edu/netlogo/.

Por otro lado se ha venido experimentando de manera lúdica y laxa con el programa *Darwin Pond*. Sus posibilidades a nivel creativo e investigativo parecen prometedoras. Se han observado procesos de homogenización en sectores aislados del hábitat virtual, lo que reflejan el surgimiento de especies en ecosistemas reales. Esto ocurre luego de varias generaciones después de que se introdujeron al ecosistema de forma deliberado *swimmers* con características aleatorias.

4 de mayo 2013

Se ha terminado de leer el manual de *Darwin Pond* al mismo tiempo que el programa está puesto en marcha. Muchas de las funciones ya se habrían intuido con la simple utilización del programa (pues es un programa realmente simple). Otras funciones fueron descubiertas durante la lectura. Una de estas, práctica e interesante, es la de la posibilidad de grabar películas en *timelapse* de los acontecimientos en el pantano en un gráfico simplificado del mismo donde los *swimmers* están representados como puntos de sus colores predominantes.

El comportamiento de los *swimmers*, el cual determina la forma en la que estos interactúan ha sido codificado en el siguiente gráfico.

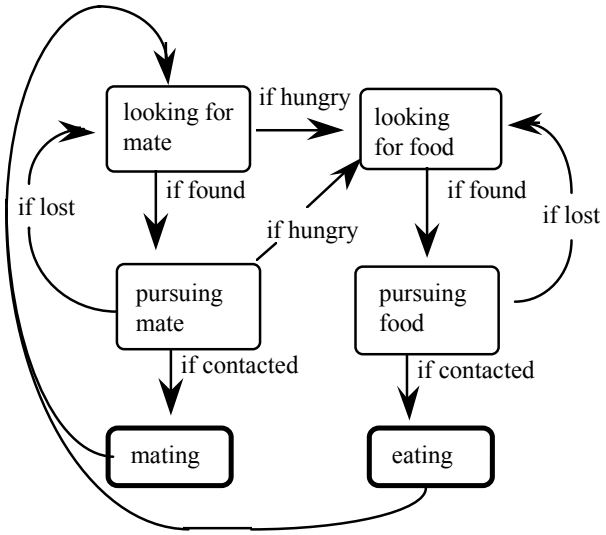


Fig. 7 Mapa de interacciones tomado del manual de *Darwin Pond* (Rocket Science Games).

Como se puede observar, las funciones vitales y el comportamiento asociado a las mismas son en realidad bastante simples.

El movimiento de los *swimmers* se basa en leyes físicas del movimiento. Las extremidades de los *swimmers* los hacen moverse en la misma manera en la que se mueven cuerpos sólidos a través de cualquier fluido. Un nadador humano sabrá que entre más grandes sean los movimientos del brazo, más líquido será desplazado empujando así al nadador más hacia adelante. Los mismos principios aplican para los *swimmers*.

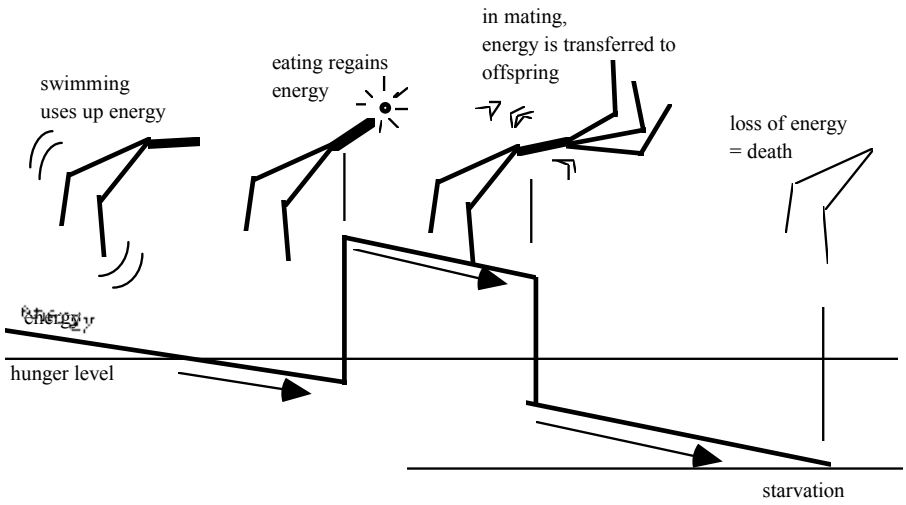


Fig. 8 Modelo de comportamiento de los *sw*immers (Rocket Science Games)

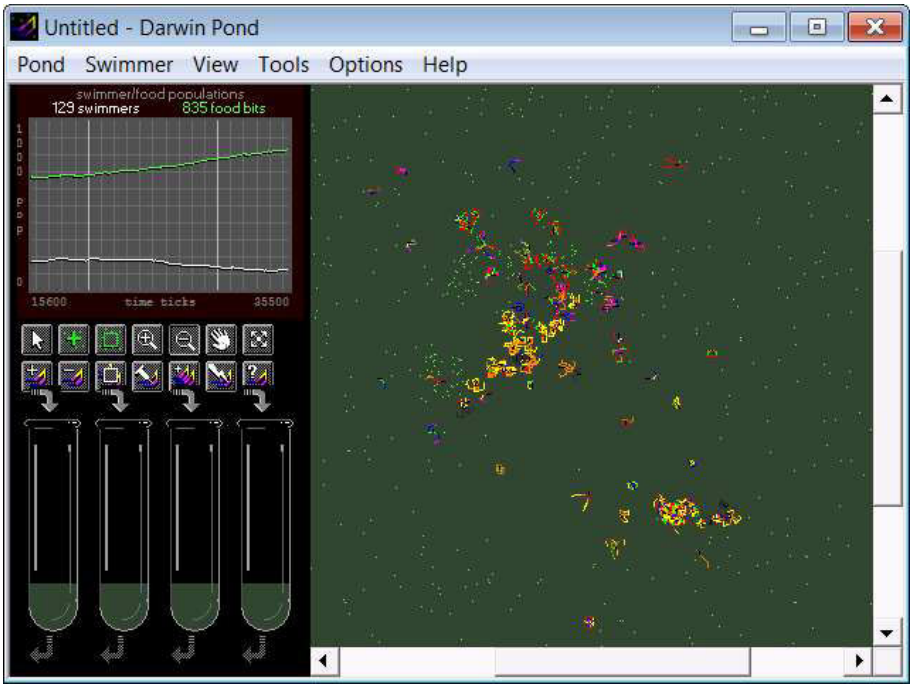


Fig. 9 Captura del programa. (Rocket Science Games)

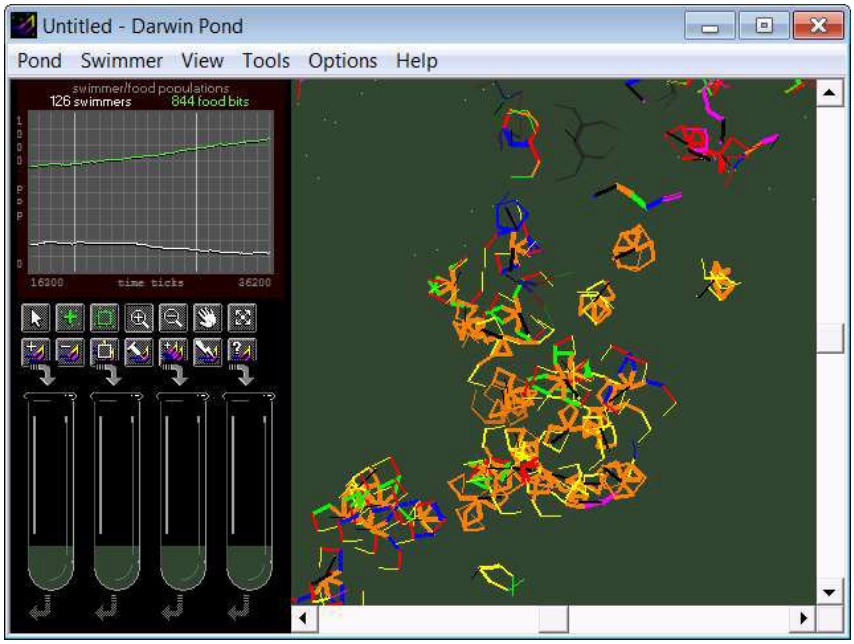


Fig. 10 La misma población vista más de cerca.

La inquietud más grande en este momento sobre la elaboración artística es la manera en la que se van a codificar las variable genéticas en términos de la creación de un material sonoro. En *Darwin Pond* se tiene acceso a un genoma de 16 genes al que fácilmente se le podrían asignar variables sonoras. El problema es entonces la manera en la que estas se expondrían como música, pues además de ser ardua la tarea de codificar los genomas de cada *swimmer* por separado, es incierto si la superposición de todos estos swimmers como motivos musicales generará o no un sonido agradable en el contexto musical. Por ahora cabe explorar otros programas y descubrir por cuenta propia si son más a menos aptos para los propósitos del proyecto artístico en la tesis.

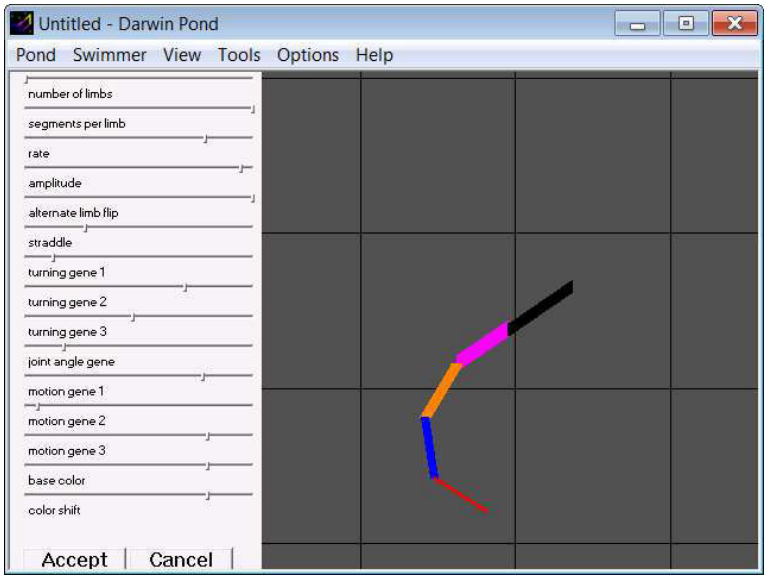


Fig. 11 Un genoma, así como su manifestación gráfica en forma de *swimmer*. Este parece una especie de serpiente.

11 de mayo 2013

Se ha descargado *NetLogo*, el programa que fue recomendado por la bióloga Lina M. Valencia.

8 de junio 2013

Ha empezado a explorarse *NetLogo*. Para informarse sobre el funcionamiento del software, el compositor del presente producto artístico ha optado por seguir los procedimientos descritos en los tutoriales disponibles en el manual de usuarios incluido en la página del programa. En el tutorial número uno, se explica la utilización en términos generales de los modelos explicada en un modelo de depredación de lobos y ovejas.

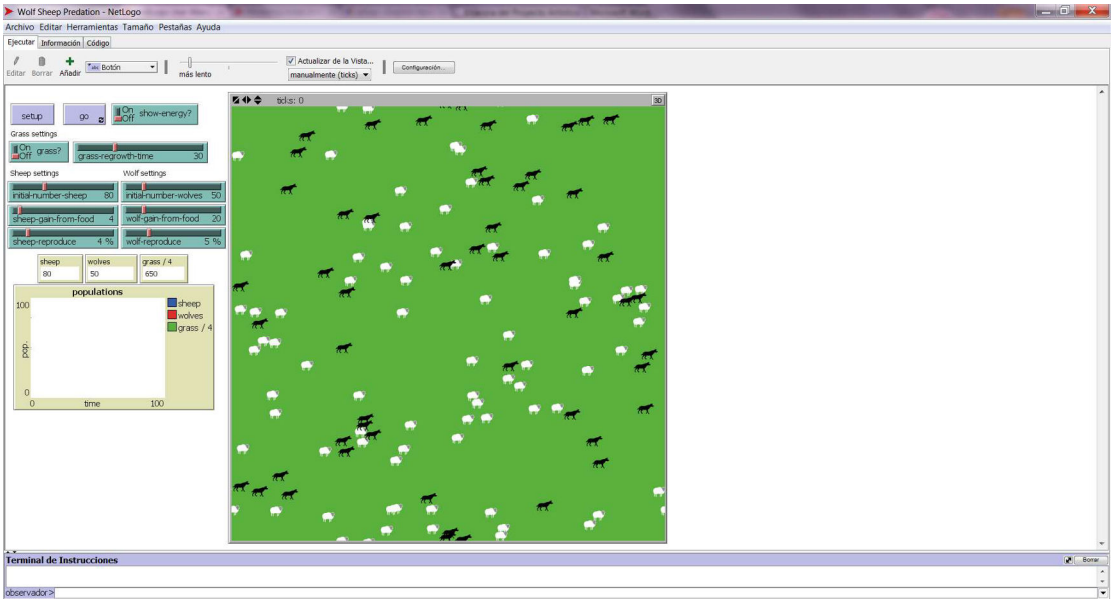


Fig. 12 Modelo *Wolf Sheep Predation* (NetLogo).

Este modelo explora la estabilidad de poblaciones en ecosistemas determinados por funciones de depredación. Un ecosistema inestable es aquel en el que se extinguen las poblaciones de individuos eventualmente. Un ecosistema estable por otro lado, es aquel que se mantiene a través del tiempo a pesar de las fluctuaciones en las poblaciones de una u otra especie.

Siguiendo los pasos expuestos en el tutorial, se descubren las funciones básicas de los modelos disponibles en el programa. Se familiariza uno entonces con las facilidades para disponer unas variables iniciales y ejecutar la partida para ver que ocurre. Así se puede estudiar el comportamiento de los agentes, denominados en la jerga del programa como *turtles* (se denominan así los agentes en este programa, independientemente de lo que pretendan representar en el modelo que se está utilizando como lobos y obejas em este caso). Este nombre para los agentes era el que se utilizaba también el programa *Logo*, del que *NetLogo* es un desarrollo. *Logo* solo permite una tortuga a la vez.

11 de junio 2013

Se han llevado a cabo los procedimientos indicados en el tutorial número dos, que se encuentra en la página de *NetLogo*. Este tutorial aborda los comandos que se utilizan para cambiar la apariencia de elementos en el sistema, en particular las tortugas o *turtles* (los agentes) y las parcelas o *patches,* que son las unidades de espacio bidimensional de las que está compuesto el suelo por el que se desplazan las tortugas, y que se identifican mediante un sistema de coordenadas. Para este tutorial se utilizó un modelo que lleva por nombre *Traffic Basic* y que estudia la formación de trancones en el tráfico vehicular tomando como variables el número de carros, la aceleración y la desaceleración.

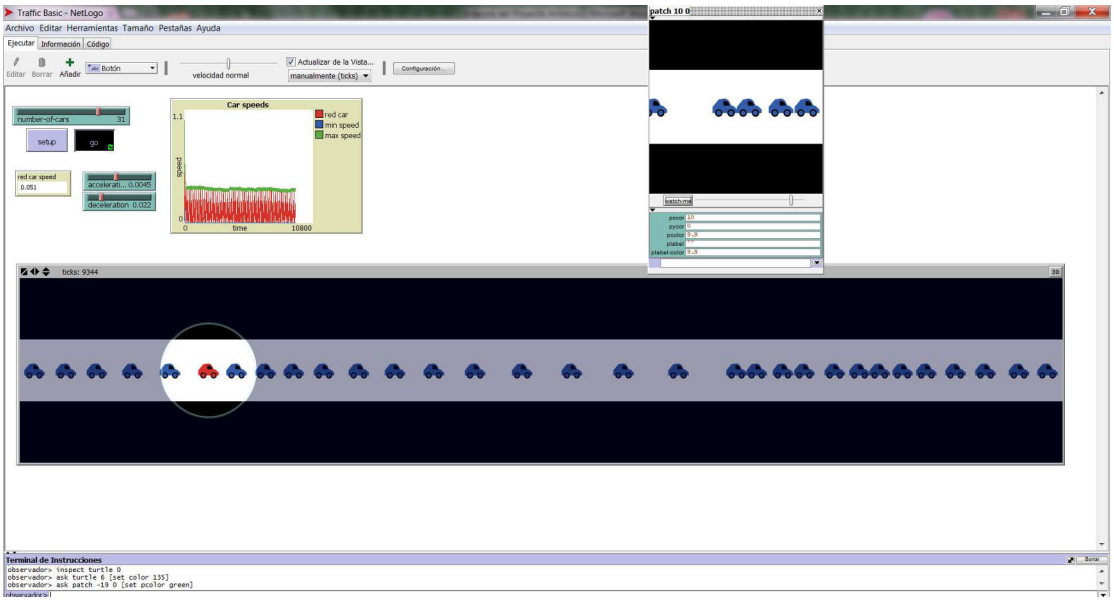


Fig. 13 Modelo en NetLogo que simula patrones de tránsito.

Los elementos en un sistema cualquiera de *NetLogo* son:

- *Turtles* o tortugas: los agentes
- *Patches* o parcelas: el terreno
- *Links* o enlaces: las relaciones entre agentes

- *Observer* u observador: entidad que monitorea lo que está ocurriendo en el sistema.

Este tutorial aborda instrucciones básicas para ser introducidas en el *Terminal de Instrucciones*, concernientes a la apariencia de los elementos en el sistema y no al comportamiento de estos (esto será abordado en el siguiente tutorial). También explica como dar instrucciones a agentes o parcelas específicas. Se expone también la manera en la que se pueden rastrear las cualidades de uno u otro agente particular en términos de color, localización, variables, figura y otros.

15 de junio 2013

Se procede entonces a realizar el tutorial número tres. Este tutorial se refiere específicamente al lenguaje de programación de *NetLogo*. Este se diligencia en forma de procedimientos en la pestaña *Código,* la cual se encuentra al lado de las pestañas *Información* y *Ejecutar.* Un procedimiento viene a ser una secuencia de comandos. Cada procedimiento empieza con la palabra *to* y termina con la palabra *end*.

El primer paso a llevarse a cabo es crear un botón, utilizando la función añadir marcada con una cruz verde en la interfaz o pestaña *Ejecutar*. Luego se le asigna un procedimiento a dicho botón en la pestaña *Código*. La siguiente es una demostración del procedimiento indicado en el tutorial.

Fig. 14 (NetLogo)

-*to setup* indica que las instrucciones han de aplicar al botón *setup*.

-El comando *clear-all* indica que se ha de reiniciar la vista removiendo todas las tortugas y todas las parcelas.

-*create-turtles 100 [setxy random-xcor random ycor]* indica que se han de crear 100 tortugas las cuales aparecerán a lo largo y ancho del tablero, puesto que los valores para *x* y *y* son aleatorios pero determinantes de un sistema básico de coordenadas.

-*end* indica el final del procedimiento.

Luego de ejecutar el código, aparece lo siguiente:

Fig. 15 Tortugas desplazándose en distintas direcciones en *NetLogo*. (NetLogo)

En la siguiente imagen se muestran procedimientos mas complejos asignados al botón *setup.* Adicionalmente se incluyeron procedimientos asignados al botón *go.* Nótese que se pueden vincular procedimientos como el del botón *go,* el cual desencadena el comando ulterior, asignado a *move-turtles.* Este controla el movimiento de las tortugas. En este caso se mueven a través de una parcela en cada tick (unidad de tiempo) estando su dirección determinada por un número aleatorio entre 0 y 29 grados (el 30 no se incluye) hacia su derecha. Este valor podría ser de 1 a 360. Si se escribiera 360 siendo este el número de grados, la tortuga podría moverse en cualquier dirección.

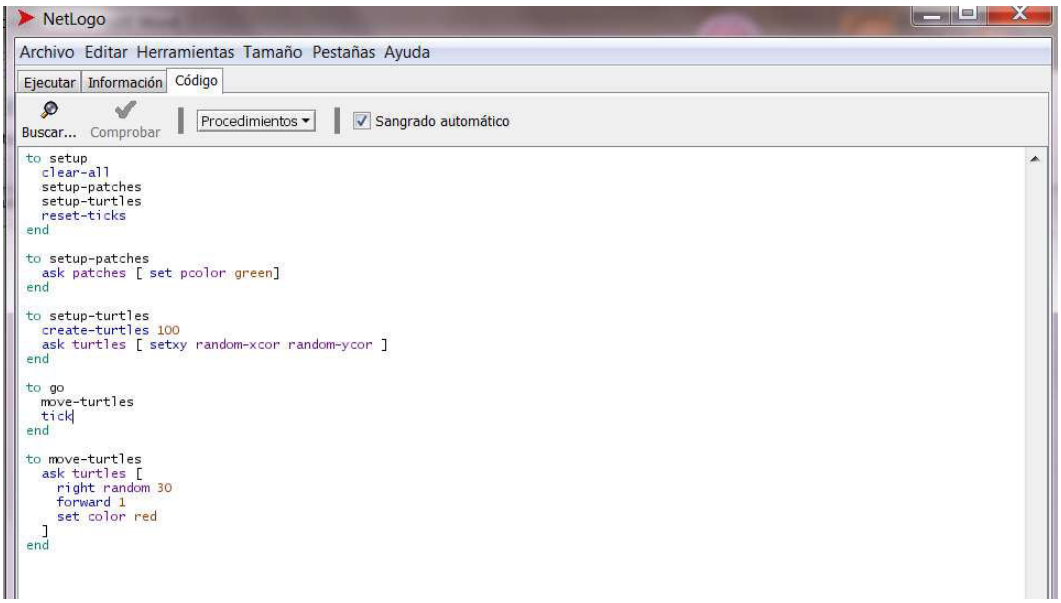


Fig. 16 (NetLogo)

Si se presiona *setup,* el resultado en la vista es el siguiente.

Fig. 17 (NetLogo)

Este sería el resultado que se obtendría si se presiona el botón *go*.

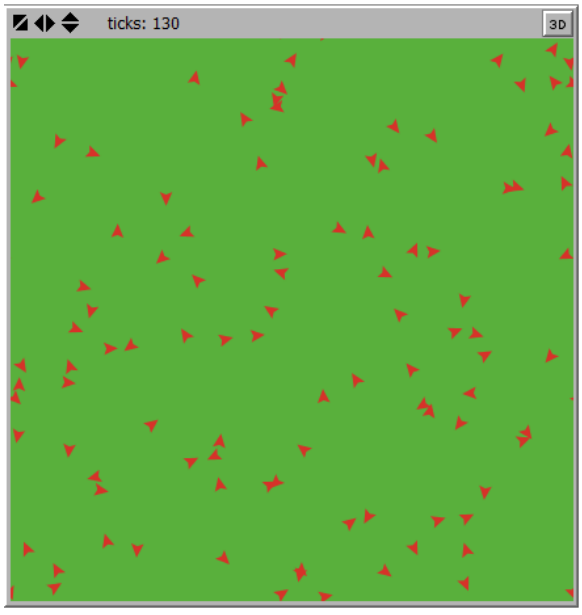


Fig. 18 (NetLogo)

Las tortugas se volvieron rojas por el comando *set color red* el cual está incluido en el procedimiento *move-turtles.*

Una función útil e interesante es la que permite el comando *pen-down*, que cuando se escribe en el panel de instrucciones hace que las tortugas vayan dejando un rastro de su movimiento sobre el terreno sobre el que se desplazan, al igual que en la versión original de *Logo*, de donde viene este programa.

Se ha realizado de manera satisfactoria la mitad de este tutorial. Prontamente se realizará completamente el mismo.

16 de junio 2013

Se procede con el tutorial número tres incluido en la página de *NetLogo.* En la parte superior del código se incluye una nueva variable: *energy.* Ahora, algunas de las funciones incluidas en el modelo estarán ligadas a los parámetros de esta variable. Propiamente, las tortugas detectaran cuando se encuentren sobre una parcela verde lo que hará que esta se vuelva negra representando el que la tortuga se ha alimentado del pasto en la misma, obteniendo así 10 unidades de energía. Cada vez que una tortuga se desplace de una parcela a otra perderá una unidad de energía. Todo esto queda diligenciado en el código de la siguiente manera.

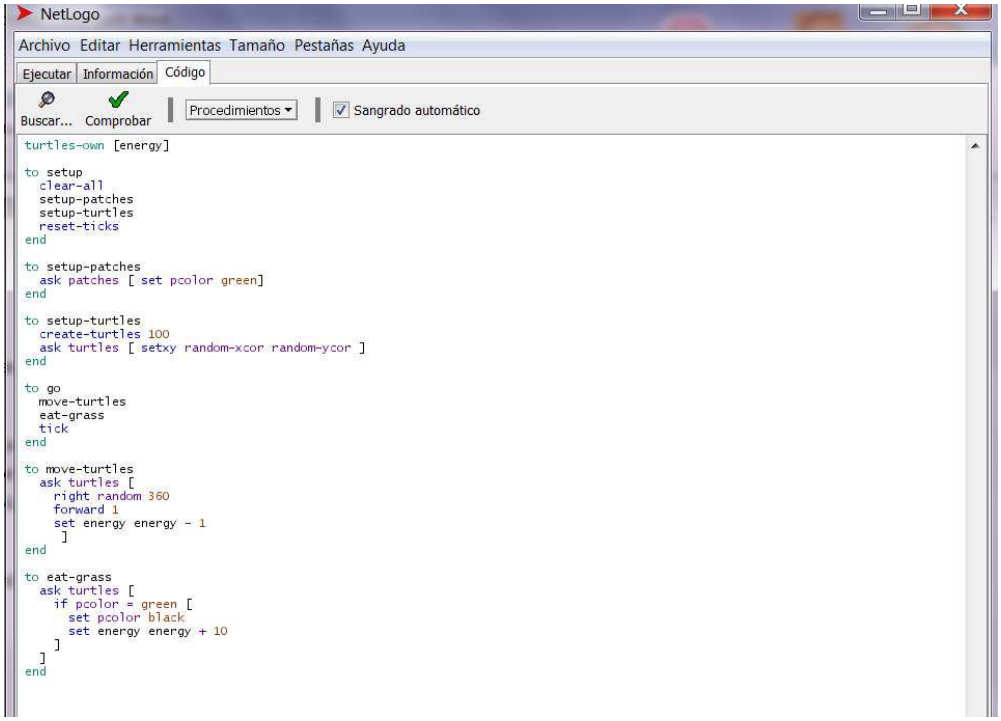


Fig. 19 (NetLogo)

Abajo se pueden ver a las tortugas luego de haber devorado un poco del pasto disponible.

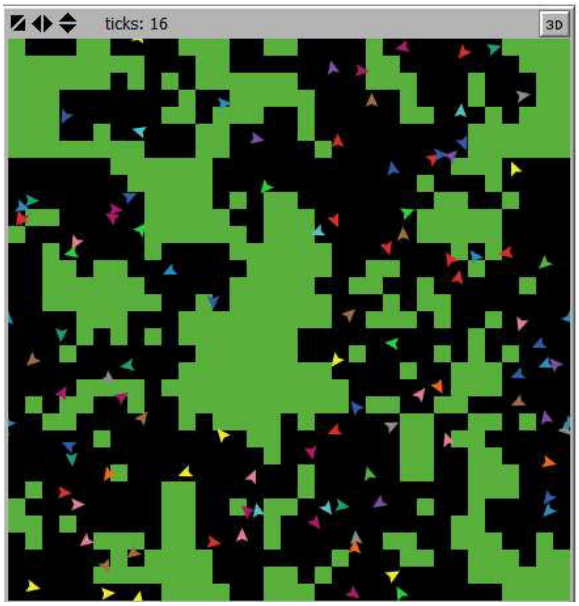


Fig. 20 (NetLogo)

Adicionalmente se incluye en el código una relación entre los niveles de energía, y la muerte y el nacimiento de las tortugas. También se incluye un procedimiento para que el pasto crezca de nuevo, todo según las indicaciones del tutorial.

Posteriormente, se le asignan al modelo monitores que marcan el número de tortugas y parcelas con pasto, un gráfico donde va quedando registrado el flujo de los números indicados en los monitores, y deslizadores que inciden directamente sobre las variables del modelo. También se le añade un interruptor para determinar si se ha de indicar el nivel de energía a un costado de cada tortuga, el cual es representado por un número blanco. Cada una de las funciones de estos nuevos botones es diligenciada con precisión en el código. Abajo puede verse el resultado de dichos procesos en la interfaz del modelo.

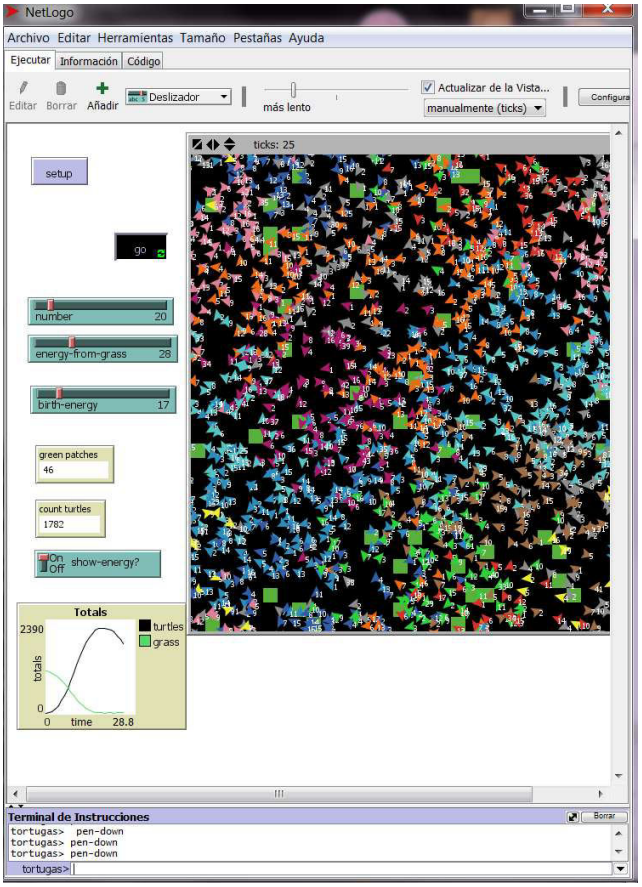


Fig. 21 (NetLogo)

Este tutorial fue el más exhaustivo de los tres ya que su propósito es dar a entender el lenguaje de programación utilizado en *NetLogo* para facilitar la creación de modelos nuevos por parte del usuario. El siguiente paso es explorar las posibilidades del programa de una manera más laxa y lúdica para familiarizarse con las posibilidades implícitas en el lenguaje de programación, al mismo tiempo que se evalúa que tan apropiado es este programa para la elaboración del producto artístico. Quedan por revisarse la guía de interfaz y la guía de programación, ambos incluidos en la página oficial. En esta última hay una lista de todos los *primitives* (comandos que realizan funciones concretas y específicas en el programa, como *hatch,* o *die*) de *NetLogo.*

11 de julio 2013

Se han explorado algunos modelos disponibles en la biblioteca de *NetLogo* relacionados con la evolución. El primer modelo explorado analiza la prevalencia de rasgos altruistas contra la de rasgos egoístas en distintas condiciones. Está basado en una competencia probabilística entre ambas tendencias. Dependiendo de las variables, le es asignada una probabilidad de prevalecer a cada una de estas. Los agentes, que en este caso son parcelas y no tortugas, representan rasgos genéticos y no a individuos.

El segundo modelo muestra la evolución del camuflaje. Este es un modelo interactivo: un juego donde la perspicacia del usuario se convierte en una variable adicional. En él, el usuario hace el papel de depredador (representado con el cursor en forma de ave), y tiene que cazar insectos que aparecen sobre una foto. Los insectos que no son cazados inmediatamente tienen la posibilidad de reproducirse pasando a su descendencia algo de sus colores en términos de pigmentos verdes, rojos y azules. Como se habría de esperar, cada vez es más difícil cazar a los insectos pues van evolucionando un camuflaje que hace más difícil su detección.

En el tercer modelo explorado, muy parecido al anterior, el usuario hace otra vez de depredador, pero esta vez la característica genética que es pasada a la descendencia de los insectos es la velocidad en la que estos se mueven. Los insectos se van moviendo entonces paulatinamente más rápido cuando se les persigue de forma activa puesto que los que pasan su descendencia son los que por ser los más rápidos son también los más difíciles de atrapar. A la inversa, si el depredador se queda esperando en un solo lugar a que los insectos lleguen a él, la característica favorecida será entonces la opuesta (la lentitud) puesto que los insectos que más rápido se mueven son más propensos a caer en

la trampa del depredador; a menos claro que esté seleccionada la opción *flee,* que hace que los insectos le huyan a la presencia del depredador.

El cuarto modelo explorado es muy parecido al modelo que estudia la prevalencia del altruismo. En este, vacas con características cooperativas compiten con vacas con características codiciosas. Las vacas cooperativas representadas en rojo solo comen una porción de pasto antes de detenerse y desplazarse a la siguiente parcela. Las codiciosas por el contrario se comen todo el pasto de la parcela en la que se encuentran antes de desplazarse, dejándola vacía para las vacas que podrían llegar a ella antes de que crezca más pasto. Las variables hacen que se vean beneficiadas más las unas o las otras. En situaciones de abundancia se reduce la discrepancia en la población de ambos tipos de vaca.

El quinto modelo aquí explorado se llama *Echo* y parecería tener potencial para la elaboración de este producto artístico. Es más una plantilla que un modelo terminado y es entonces posible y aconsejable enriquecerlo diligenciando material nuevo en el código para hacerlo más complejo y más realista (en términos de ecosistemas reales). En él, las tortugas se relacionan unas con otras en base a variables que aplican a todas ellas y a un código genético muy sencillo donde interactúan tres variables esenciales. La primera es *ataque* (a), la segunda es *defensa* (b) y la tercera *apareamiento* (c). Hay entonces un paradigma de depredación que no existe en *DarwinPond,* puesto que las variables genéticas de ataque y defensa determinan la manera en la que los agentes toman energía unos de otros. También pueden obtener energía de las parcelas, lo que constituiría un primer piso trófico análogo al nicho que ocupan plantas en ecosistemas reales.

Resulta preocupante, no obstante, lo simple que es el genoma de cada agente. Con su reducido número de variables quedaría entonces gran parte de la riqueza musical limitada a las interacciones entre los agentes, los cuales estarían codificados como motivos sonoros. Adicionalmente, cada uno de los genes tiene funciones que se limitan a una seria corta de valores numéricos.

Tres variables sonoras que se podrían codificar en base a estos sencillos genomas, mediante analogías subjetivas, serían duración, frecuencia y amplitud. La duración podría entenderse como segmentos rítmicos impuestos a una medida de tiempo estándar ya que de otra manera podrían chocar con el registro evolutivo a través de las generaciones, entendido este en la composición musical como el paso del tiempo. Las variables tímbricas podrían estar asociadas al color de las tortugas. Irían surgiendo estas como patrones de especiación, los cuales permite el modelo. Una gran ventaja de utilizar *NetLogo* en vez de *DarwinPond* es que el primero permite avanzar a través del proceso evolutivo un *tick* de tiempo a la vez. Esto facilitaría el registro de las variables emergentes para su subsiguiente codificación sonora.

En los modelos de muestra, en la pestaña del código, está escrita en gris una descripción de lo que significan los comandos y procedimientos.

13 de julio de 2013

Se han explorado modelos adicionales que imitan comportamientos evolutivos. Los primeros cuatro en explorarse en esta sesión exploran el concepto de la deriva genética

(*genetic drift*). Este concepto infiere que la emergencia de muchos alelos ocurre en gran medida por una recombinación azarosa del material genético que no está sujeta a presiones de selección natural. Un alelo es una de las posibles manifestaciones de un *gen* o un grupo de genes. Por ejemplo, el gen que determina el color de los ojos en un ser humano se divide en alelos que hacen que este sea o café, o verde, o azul, etc. La diferencia entre la evolución determinada por deriva genética y la guiada por la selección natural es que la primera es un proceso azaroso mientras que la segunda encamina el proceso en base a rasgos adaptativos que facilitan la supervivencia del individuo (no es azarosa, por lo menos no del todo). Hoy en día se ha llegado al consenso de que ambas fuerzas tienen influencia en el proceso evolutivo teniendo más significancia la una o la otra dependiendo de las circunstancias.

Los modelos de deriva genética son mucho más aleatorios que los otros en la biblioteca pues no hay variables que simulen las presiones de la selección natural o artificial. En la primera, las parcelas imitan el color de cualquiera de las otras parcelas hasta que una prevalece. El siguiente modelo es como este pero las parcelas imitan solo una de las ocho parcelas en su periferia. El tercer modelo hace que las tortugas adopten el color de las tortugas con las que entran en contacto. La última explora la deriva genética en un modelo de reproducción sexual. Al igual que las otras tres, un color prevalece sobre los demás.

Se exploraron otros tres modelos. El primero explora el fenómeno etológico de la *mímesis*, donde una especie no peligrosa imita el aspecto de una que sí lo es para evitar ser comida. En él, coexisten tres agentes: mariposas monarca, las cuales tienen un desagradable sabor; polillas viceroy, las cuales imitan la apariencia de las mariposas

monarca para evitar ser comidas; y pájaros, que son los depredadores que se alimentan de ellas.

El siguiente modelo simula un ejemplo que frecuentemente se utiliza para explicar la selección natural. En los bosques de Manchester existían polillas blancas especialmente adaptadas para camuflarse en la también blanca corteza de los arboles sobre los que se posaban. Cuando llegó la revolución industrial, el tizne que emanaba de las fábricas cambió el color de los árboles dejando más expuestas a las polillas blancas y haciendo más inconspicuas a las oscuras. Es por eso que estas últimas prevalecieron. Este estudio etológico ha sido reevaluado desde entonces y se ha concluido que hubo inconsistencias en el método deductivo. Las implicaciones generales del mismo, no obstante, son ampliamente aceptadas en la comunidad científica.

Las variables más importantes en este estudio son el color de las polillas, el nivel de contaminación, y una variable denominada selección, que es un indicador general de las presiones evolutivas (que tantos pájaros hay, que tanta hambre tienen, etc.). Las mariposas más oscuras son predominantes en la población cuando ha aumentado el nivel de polución y las blancas lo son cuando los niveles de polución son bajos.

El último modelo evolutivo en explorarse fue una versión más pintoresca del *Relojero Ciego* de Richard Dawkins, del cual ya se ha hablado en la investigación.

Se considera que tal vez *NetLogo* no sea la mejor opción para la elaboración del proyecto artístico. La razón principal de esta conclusión es que sus modelos evolutivos están diseñados para simular fenómenos aislados dentro de la ciencia evolutiva y no el proceso evolutivo *per se*, que es lo que se desea para el proyecto. Es necesario un

simulador donde pueda ocurrir el proceso evolutivo de abajo para arriba (es decir, de simple a complejo, o de aleatorio a organizado como en *DarwinPond*) y en este programa esa posibilidad no parece viable.

Desde luego que no se descarta por completo la posibilidad utilizarlo. Se podría, por ejemplo, diseñar con el código un modelo que haga precisamente lo que se necesita. Adicionalmente, explorar el programa otorgó a la investigación grandes entendimientos sobre evolución y sistemas, y las posibilidades musicales del programa en un contexto por fuera del presente, son potencialmente interesantes y virtualmente infinitas.

13 de febrero 2014

Hoy se exploraron las posibilidades del programa *Framsticks Theater*. Es esta una versión de *Framsticks* en la que se proyectan, entre otras cosas, muestras programadas de simulaciones evolutivas básicas de selección artificial tipo *The Blind Watchmaker*, y carreras de organismos. El paquete también trae consigo el software para hacer la programación de *Framsticks* en código uno mismo. En general no obstante, aunque el programa es muy interesante y tiene potencial como fuente de valores que podrían adaptarse a muchos tipos de composición musical, no es realmente viable para el actual proyecto.

14 de febrero 2014

Esta mañana fue descargado un programa de nombre *Biogenesis*. El programa utiliza una interfaz simple y es fácil de instalar. En él, unos organismos construidos a base de segmentos (en esto se parecen a los de *DarwinPond*), evolucionan en un ambiente de

dos dimensiones donde hay fenómenos de depredación, fotosíntesis, y otros más interesantes como infecciones, en las que un organismo fuerza a otro a tener su cría (osea, que surge como manifestación de los genes del "infectador" y no como manifestación los del huésped infectado). Esto lo hace distinto a *DarwinPond* donde la única habilidad que determina el *fitness* de los organismos es la habilidad para desplazarse nadando. Los organismos en Biogenesis no solo se desplazan también, sino que tienen segmentos de una variedad específica que les permite desplazarse distancias mayores y a mayor velocidad.

Una cosa que *DarwinPond* tiene pero *Biogenesis* no, es la posibilidad de recombinación genética tipo reproducción sexual. Todas las criaturas en *Biogenesis* son asexuales y sus descendientes son versiones mutantes de ellas (el grado de divergencia en el genotipo y el fenotipo de la criatura descendiente puede graduarse en términos probabilísticos alterando el factor *mutación* en los parámetros del programa). Esto no impide que haya evolución por selección natural. Tampoco impide que en el surgimiento de linajes altamente idiosincráticos se pueda vislumbrar, análogo fenómeno especiación.

Biogenesis es el programa más apto para este proyecto de entre todos los que se han explorado hasta ahora. Es entonces el programa que se utilizará en su elaboración. Se han tomado las siguientes decisiones (sujetas a cambios solo si es verdaderamente necesario).

- La obra se presentará como una *instalación cuadrafónica,* con un parlante en cada esquina de un área rectangular. Con esto se expresará la posición del organismo en el susodicho plano bidimensional utilizando principios de espacialidad.
- Un video del desarrollo de la evolución de donde se ha modelado la composición será proyectado a la par de la música para vincular al público con el proceso de generación de esta en una experiencia multimediática.
- La masa del organismo será inversamente proporcional a su frecuencia (basado en la lógica de que los objetos grandes emiten sonidos más graves que los pequeños).

24 de Febrero 2014

Las siguientes son opciones que se han contemplado para la codificación de organismos como paquetes de síntesis.

Conjunto de opciones No. 1

No es simultáneo (es decir que la representación del organismo existe como un evento en el tiempo y por lo tanto contiene implicaciones rítmicas. Ante todo los segmentos involucrados se reproducen consecutivamente en lugar de en simultanea).
Longitud de segmento: décimas de segundo (los decimales representan centésimas)
Simetría: división de la octava.
Rotación: Distancia en la frecuencia entre un segmento y el anterior. (Intervalo máximo: Octava = 180°. Rotación positiva: se divide la octava por 180 y se suma a la frecuencia anterior. Rotación negativa: se divide la octava por 180 y se resta de la frecuencia anterior).

Conjunto de opciones No. 2

Todo igual al conjunto 1 pero simultaneo (las representaciones de los segmentos están solapadas, ósea que las frecuencias suenan al mismo tiempo).

Conjunto de opciones No. 3

Como el conjunto 1 pero el valor de simetría representa el número de octavas donde se repiten las relaciones entre segmentos (un organismo con una simetría de 2 estaría entonces una vez octavado).

Conjunto de opciones No. 4

Como el conjunto 2, pero con la aplicación en distintas octavas que se le hace en el

Conclusiones sobre la implementación de estas opciones:

Conjunto de opciones No. 1

El organismo con el que se experimentó era muy simple y no obstante su resultado sonoro fue demasiado complejo (y demasiado estridente también). Es ahora menester formular métodos para codificar los organismos para que tengan resultados sonoros que sean proporcionales al nivel de complejidad de los organismos en su representación visual.

Conjunto de opciones No. 2

Los cálculos que se hicieron con respecto a las frecuencias en hercios no cuadran con el intervalo que se esperaba. Al dividir los hercios de la octava en dos se esperaba conseguir un tritono pero el resultado es mucho más parecido a una quinta justa. Se ha conseguido una tabla de correspondencias entre las notas del sistema temperado y el número de hercios que las componen en la afinación La 440. Las correspondencias entre frecuencias y alturas son menos parsimoniosas de lo que alguna vez se comprendió.

Segundo experimento para la elección de métodos de codificación de organismos:

Conjunto de opciones No. 5

Cada orden de simetría adicional añade un nuevo oscilador con una frecuencia igual a la original más la mitad de la misma.

X2= X1 + (1/2*X1)

El ángulo de rotación determina la inclinación en el envolvente. La frecuencia donde termina está determinada por la operación trigonométrica dados los valores del ángulo y la longitud del segmento. Cada 2° del ángulo del segmento equivalen a 1° del ángulo en el envolvente con respecto al segmento anterior.

Nota sobre el proyecto:

Toda la amplitud será proporcional. Cada fila de segmentos en un organismo con tres de estos tendrá entonces 1/3 de la amplitud total de todo el organismo. Si dos segmentos en el tentáculo tienen 18 pixeles de longitud, por ejemplo, la proporción sería entonces 18:18 que es igual a 1:1, osea que un tercio de la amplitud total del organismo se divide en dos con lo que cada segmento obtiene una sexta parte de la amplitud disponible. Otras proporciones determinarían la proporción de amplitud entre los sonidos dentro de una misma fila de segmentos, que es a su vez una fracción de la amplitud de todo el organismo, y esto depende del valor de simetría.

Con esto se busca que los organismos más complejos (es decir, con mayor número de genes), no suenen necesariamente a un nivel más alto que los más simples.

Ideas para la aplicación del valor de simetría:

-Sobre segmentos verdes: Número de alturas en un acorde por quintas.

-Sobre segmentos azules: Número de batimientos por lapso de tiempo en onda cuadrada mediante el uso de síntesis modular.

-El uso del ángulo de rotación entre segmentos es igual al del conjunto anterior.

Conjunto de opciones No. 6

El valor de simetría corresponde a íntegros de la frecuencia base añadidos sobre la misma como ocurre en la serie armónica.

El ángulo de rotación determina, entendiendo la longitud del segmento como la hipotenusa de un triángulo de ángulo recto, cuantos hercios va a subir la frecuencia (opuesto) y cuanto se va a demorar haciéndolo (adyacente).

Nota adicional:

La síntesis modular puede utilizarse para:

-Crear timbres específicos para representar los colores de los segmentos.

- Generar batimientos cuyo número de iteraciones por lapso de tiempo represente el valor de simetría de cada organismo.

27 de febrero 2014

Se ha considerado la opción de cambiar la metodología del proyecto. Donde originalmente se planteó la composición como una transcripción de valores genéticos modelando una evolución en *Biogenesis*, ahora se considera la posibilidad de volver esta una obra generativa no determinista donde se sincronizarían los valores de *Biogenesis* con los de un programa de síntesis sonora en tiempo real.

Se han evaluado los pros y los contras de ambas opciones.

Modelaje del proceso evolutivo

Pros:

- Se ha conceptualizado mejor que la otra opción.
- Se pueden hacer varias transcripciones (suponiendo que haya tiempo suficiente) y elegirse la mejor para la presentación del proyecto.

Contras:

- Toma demasiado tiempo.
- Posee menor dinamismo al estar su reproducción completamente determinada.

Música generativa en tiempo real

Pros:

- En el espíritu del concepto de la evolución, la música generativa daría una sensación más orgánica, entre otras razones porque la evolución biológica en sí un proceso solo parcialmente determinista.
- A la larga requiere menos tiempo y trabajo manual.
- Otorga la posibilidad de generar música nueva ilimitada.
- Trabaja con un enfoque más contemporáneo en términos de las pretensiones del concepto a realizarse (o puesto de otra manera, es más congruente al contexto actual).

Contras:

- No se ha planeado aún en detalle.
- Requiere experiencia con computadores.

Se explorarán más a fondo *Supercollider*, *Max MSP*, *PureData* y *Python* (un programa que sincroniza programas distintos) para considerar mejor este posible camino.

19 de Marzo 2014

Hoy se asistió a la antepenúltima sesión del taller de *Max MSP* dictado por el compositor Argentino y miembro de la CMMAS, Francisco Colasanto. Se ha obtenido entonces mayor entendimiento y familiaridad con el lenguaje del programa y las técnicas de programación que permite. Todavía persiste el grueso de la incógnita sobre cómo se desarrollará la opción generativa en la composición. Se le preguntó a Francisco si es posible conectar Java (*Biogenesis* está escrito en java) con *Max*. Su respuesta fue afirmativa, y demostró un objeto específico para Java (el objeto [mxj]) y otro para *JavaScript* (el objeto [js]). El prospecto es ahora alentador.

El siguiente paso es aprender Java. Ya se han localizado tutoriales de este lenguaje en Internet. También toca aplicar todo lo que se aprenda sobre Java al propósito de la extracción de variables de *Biogenesis* para ser introducidas estas en un patch de *Max MSP*. Podría llegar a resultar problemática la gran cantidad de valores arrojados por el código de una evolución de *Biogenesis*.

25 de marzo 2014

El viernes pasado fue la última sesión del taller de M*ax MSP*. En el taller, el compositor del presente proyecto se familiarizó con el lenguaje y las técnicas del programa. Este se dirigió al final de la sesión a donde Colasanto y le habló sobre la presente composición. Colasanto sugirió que es mejor utilizar *Osc* para este proyecto en lugar de *MIDI,* el cual, según él, es menos viable. Habló de dos objetos: el objeto [id], que se le puede asignar a paquetes de síntesis aditiva particulares (que en este caso representarían a los organismos), y el objeto *flag*, que equivale a los parámetros de dichos paquetes.

Adicionalmente se ha comenzado a seguir un tutorial de Java que está en *YouTube* (https://www.youtube.com/playlist?list=PLE7E8B7F4856C9B19). El tutorial consta de muchos videos por lo que es importante avanzar a través de ellos con celeridad, pues el tiempo disponible para la elaboración del proyecto es una preocupación importante a estas alturas.

Adicionalmente, Ana María Romano, la asesora específica del presente proyecto, comentó que tiene una hermana que sabe trabajar con Java y que tal vez pueda prestar algún tipo de asesoría.

También planteó la posibilidad de hacer un encuentro vía *Skype* con un compositor colombiano que vive en Europa y que ha trabajado con similares conceptos. Acá se podrán aclarar inquietudes con respecto a la forma más viable de hacer la programación para el proyecto.

Se contempla el que Java pueda arrojar variables para que estas sean tomadas por los objetos [id] en *Max MSP* interpretados como *flags* para su subsecuente reproducción como información dirigida a osciladores pre existentes (latentes), o generados nuevos a partir de un código que permita hacer tal cosa.

30 de marzo de 2014

En este fin de semana que termina se hicieron avances sustanciales. Se ha estado utilizando *Max MSP* de forma más lúdica para así familiarizarse con el programa. El tiempo acumulado utilizando el programa facilitará la eventual elaboración del *patcher* para el proyecto. Adicionalmente se están haciendo un tutoriales de *Max MSP* en YouTube (https://www.youtube.com/playlist?list=PL3E0F7EB6BDD433DF). Ya se han realizado cuatro tutoriales de esta lista.

Sobre la concepción del proyecto se han determinado conceptos de modulación para cada color de segmento de los organismos, así como para los estados de los organismos cuando atacan a otros, infectan a otros, se defienden de los ataques de otros, o padecen los ataques de otros. También se han conceptualizado posibles aplicaciones para el valor de energía.

Se ha aprendido sobre la utilización de *sub-patchers.* Estos son algoritmos con distintos grados de complejidad, que están contenidos dentro de objetos de Max. Facilitan la organización y la limpieza de los *patchers.* Existe un comando denominado "Encapsular" para tomar cualquier selección de un *patcher* y encapsularla dentro de uno de estos objetos.

Se han hecho numerosos bocetos que poco a poco van dilucidando problemas técnicos sobre el proceso. Hoy por hoy, la incógnita principal es como hacer sonar distintos módulos de síntesis aditiva al mismo tiempo. El objeto [poly~] podría ser de utilidad para este propósito.

8 de abril de 2014

Se han diseñado ya los timbres que se le asignarán a cada uno de los segmentos de los organismos. A los segmentos verdes se les darán simples ondas sinusoidales. A los rojos, ondas diente de sierra. A los azules ondas cuadradas. Los cyan serán ondas sinusoidales con un vibrato producido mediante la modulación de la frecuencia con otra onda sinusoidal. Los amarillos serán ondas sinusoidales con el mismo tipo de modulación que los cyan, pero con un ancho de banda mayor y una frecuencia menor, produciendo así un vibrato más amplio pero más lento. Los segmentos blancos tienen una onda moduladora de amplitud cuya amplitud es a su vez modulada por otra, que es a su vez modulada por otra más, también en su amplitud. El sonido es un sonido caótico y más o menos estridente, con el que se busca dar la sensación de que algo está infectado, pues este segmento sirve para que unos organismos infecten a otros con su propio material genético. Los segmentos grises tienen un sonido más estridente aún puesto que son segmentos que matan a los otros organismos cuando estos entran en contacto con ellos.

Así mismo, cada timbre se escogió teniendo en cuenta las funciones de los segmentos de cada color. La onda sinusoidal se le otorgó a los segmentos verdes porque tienden a ser los más ubicuos y porque siendo responsables del proceso de fotosíntesis,

representan en la sencilla pirámide alimenticia el piso trófico base, el más elemental. Igualmente, la onda sinusoidal podría considerarse la más elemental de las ondas puras en el ámbito de la acústica. La onda diente de sierra representa la cualidad hostil de los segmentos rojos que los organismos utilizan para la depredación. La onda cuadrada llena el espacio armónico de tal manera que la hace propicia para la representación de la cualidad defensiva de los segmentos azules. La onda con modulación lenta y amplia representa el hecho de que los segmentos amarillos le otorguen al organismo una ventaja reproductiva (mayor número de crías). Para este efecto se busco un sonido que diera la sensación de voluptuosidad. El vibrato rápido de los segmentos cyan representa el que estos ayuden a los organismos que los poseen a desplazarse como los cilios (especie de látigos ondulantes) ayudan al desplazamiento de algunas células reales, como por ejemplo los espermatozoides.

El comportamiento musical de los organismos se determinará en distintos niveles, los cuales pueden estar estructurados con un sistema de *sub patchers*. En el nivel más pequeño estarán determinados la suma de los timbres representativos de los segmentos así como las proporciones entre sus amplitudes y sus frecuencias, que están asociadas respectivamente a las longitudes de los segmentos y a los ángulos de rotación entre ellos.

En un nivel superior se modulará todo el módulo del nivel inferior mediante valores determinados por los valores de simetría y de simetría espejo en el código genético de los organismos. Esto determinará la frecuencia de iteraciones o pulsaciones rítmicas. Este efecto se logrará mediante la modulación de las amplitudes de todas las ondas representativas de los segmentos en conjunto.

En un nivel superior, o varios de estos, dependiendo de lo que resulte más práctico, se colocarán multiplicadores que afecten la amplitud y frecuencias generales de todo el organismo para cada uno de ellos. Amplitud y frecuencia están respectivamente asociadas a la masa y a la energía de estos. También se pondrán en este nivel valores referentes al eje *x* y al eje *y* para determinar así la posición de cada organismo como un objeto sonoro en el espacio cuadrafónico.

El problema en concepto que aún persiste concierne a la creación espontanea de los organismos que van naciendo. Una solución puede hallarse en la inclusión de un número limitado de oscilados que no aparecerán a la par de los organismos, sino que serán activados por estos estando latentes antes de que esto ocurra.

La aparición y existencia prolongada de un organismo puede darse como afirmación sintáctica (*boolean*) para abrir los canales entre los osciladores latentes y la fuente de sonido al mismo tiempo que insufla a los módulos con variables provenientes de clases java para concederles sus cualidad idiosincráticas.

Otra solución puede ser tener solo algunos cuantos osciladores con varios canales que se puedan abrir por las variables provenientes de java resultando en una manifestación polifónica de estos pocos generadores de tono. Todas estas consideraciones son tentativas, por lo menos parcialmente.

13 de abril 2014

Se ha creado un organismo prototipo en *Max MSP,* sobre el cual se seguirá elaborando para que así se vallan resolviendo consideraciones técnicas en el proceso. El organismo es un modulo sencillo de síntesis aditiva donde valores se le aplican a la frecuencia base de las ondas del organismo así como a la suma que tendrán los otros tipos de ondas sobre esta frecuencia base, lo que representará el ángulo de rotación en los organismos de *Biogenesis*. Los valores arrojados por los generadores de números aleatorios serán remplazados por valores arrojados desde los objetos java. No se le han aplicado valores de amplitud y de amplitud relativa aún al modulo. Esto queda pendiente y su elaboración necesitará reconocer números decimales, o convertir íntegros en decimales.

Se ha escrito una entrada en el foro de *Biogenesis* donde se explica la naturaleza del presente proyecto, y donde se pide explícitamente información sobre la localización de aquellas variables que afectaran la calidad de cada organismo en la composición. No solo está incógnita fue respondida por el usuario Tyler, sino que además este se ofreció a ayudar con otros aspectos técnicos de la programación de la obra.

Se atendió a unos tutoriales de *YouTube* (https://www.youtube.com/watch?v=LVFPCb6KsrE) que explican específicamente la vinculación de Java con *Max MSP* por medio del objeto [js]. Se aprendió a declarar funciones sencillas en java que convierten información introducida en el objeto para arrojarlos por los *outlets*. Para esto toca declarar no solo las funciones, sino también los *inputs* y los *outputs*, todo utilizando esto en JavaScript. Algunas preguntas a responder ahora son:

-¿Tienen que estar los archivos de java que se usan en Max especialmente diseñados para Max?

-¿Puede sostener el objeto [js] archivos complejos de java con clases y subclases, como los de *Biogenesis*?

-¿Puede programarse en java un objeto [js] que tenga un funcionamiento interno aislado al de Max en cuanto a la entrada o el procesamiento de sus datos, pero que tenga outlets declarados por los que se arrojan algunas de sus variables a otros objetos en Max?

-¿Puede existir una manifestación visual al tiempo que una sonora (la visual siendo el mundo donde viven los organismos de *Biogenesis*), y pueden las dos reproducirse al tiempo para permitir el aspecto multimediático de la composición?

Estas preguntas y otras pueden ser formuladas en los foros de *Max MSP*.

18 de abril 2014

Se ha elaborado sobre el diseño del patch *ProtoOrganismo.maxpat*, el organismo prototípico previamente mencionado. Se les dio a los valores globales de frecuencia y amplitud modulación por una onda de baja frecuencia que hace que la frecuencia y amplitud global no idiosincrática del organismo sea un portamento similar al que ocurrirá en el proyecto actual. Los valores de frecuencia y amplitud relativa que marcarán la idiosincrasia (el conjunto de características específicas) de cada organismo particular en la versión final del proyecto, se le asignan a este cada 34 segundos a partir de generadores de números aleatorios.

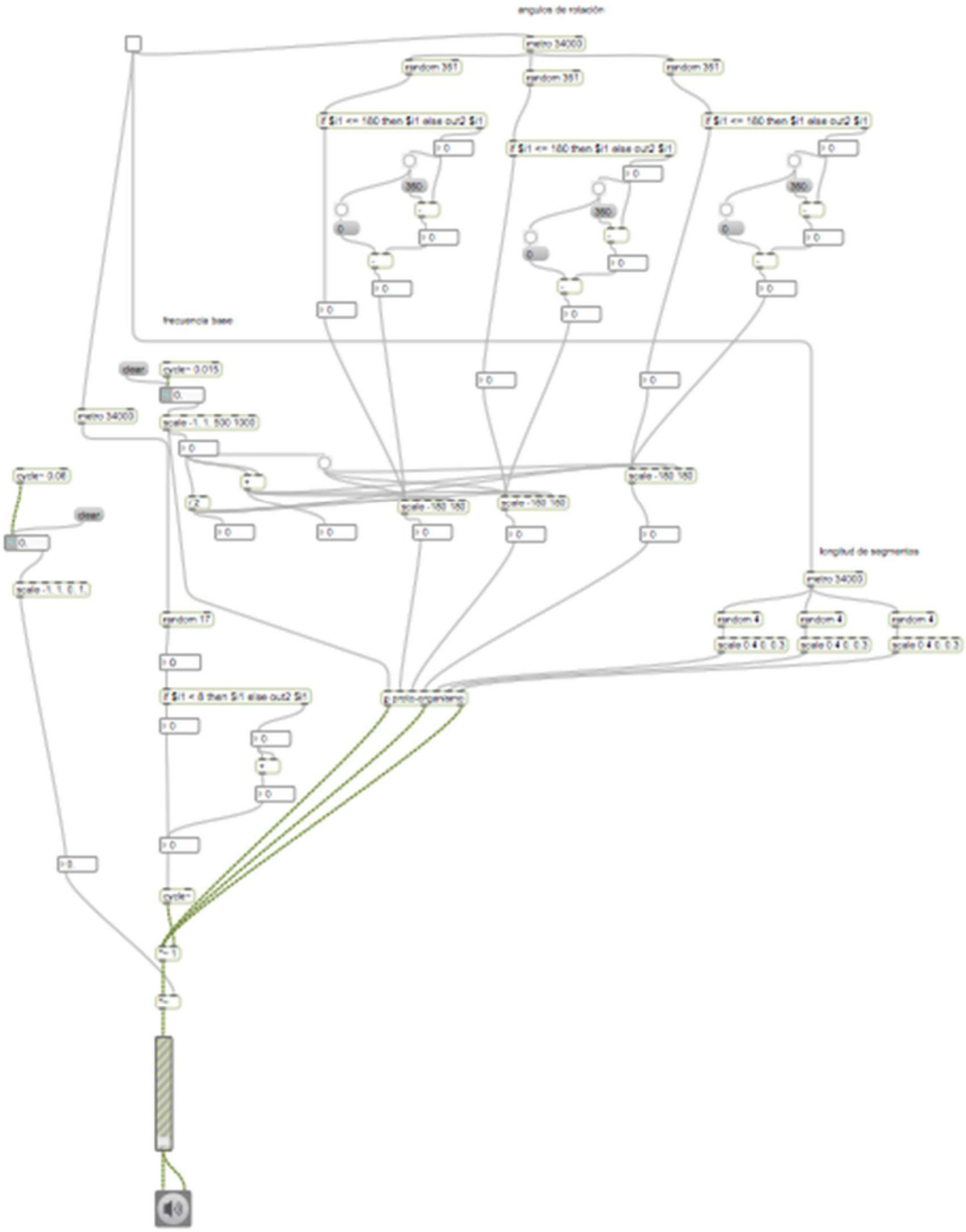


Fig. 22 *ProtoOrganismo.maxpat*

Se han yuxtapuesto distintos organismos de este tipo para hacerse una idea del efecto polifónico que se conseguirá en el proyecto actual. Los valores de frecuencia de las ondas de baja frecuencia que se mencionaron previamente están alterados sutilmente con respecto unos de otros para generar un comportamiento dinámico particular para cada organismo, permitiendo entonces distinguirlos entre sí.

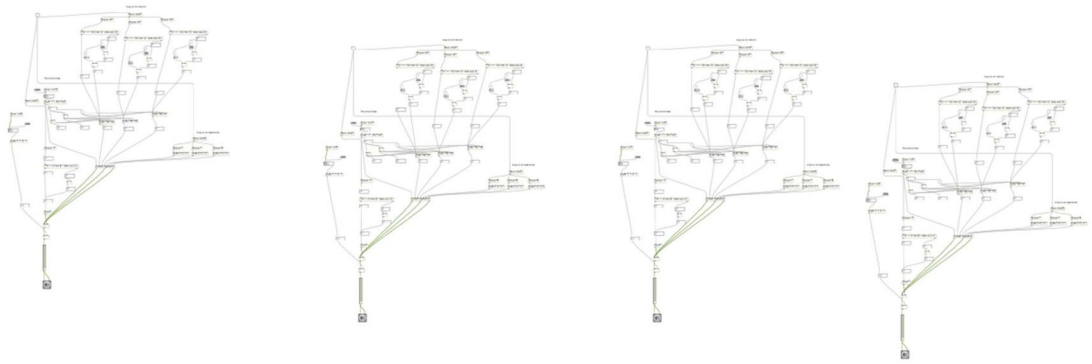


Fig. 23 ProtoOrganismo.maxpat, visión de pájaro.

Este patch no contiene entre sus variables las que refieren a un comportamiento espacial. Queda entonces pendiente la construcción de un sistema que incluya todo lo que contienen estos proto-organismos, más las operaciones sobre variables de horizontalidad y verticalidad en el plano bidimensional donde viven. Queda tambien pendiente todo lo referente a la extracción de variables en Java, y el problema sobre las operaciones de polifonía globales y generación de fenotipos sonoros para organismos emergentes.

3 de mayo 2014

Se han hecho avances significativos en cuanto al uso de foros para recibir retroalimentación por parte de usuarios de los programas que estoy utilizando. Se

ingresó al foro de *Biogenesis* donde el usuario Tyler aconsejó seguir un tutorial cuyo vínculo incluyó en su entrada. Se llevó a cabo el tutorial en cuestión con lo que se consiguió mandar variables de una clase externa de java por medio del objeto [mxj] y el programa *Eclipse* donde se encontraba la clase. Se procedió entonces a copiarse el código de *Biogenesis* en el proyecto que fue creado en eclipse para vincular *Eclipse* con *Max MSP*.

Se está buscando asesoría en programación por parte de un técnico acá en Bogotá, pues no parece viable aprender los detalles del lenguaje de programación Java con el poco tiempo que se tiene para la culminación del presente proyecto. La tarea de conectar *Biogenesis* con Max, permitiendo la sincronía, es decir, la ejecución simultanea de ambos programas mientras se mandan datos de uno a otro, sería más sencilla si se cuenta con el apoyo de alguien con experiencia en el tema.

8 de mayo 2014

No se ha conseguido aún un técnico con experiencia en Java a pesar de que se han intentado contactar cinco distintos. Por otra parte, con la información provista por Tyler en el foro de *Biogenesis*, se ha logrado ejecutar el programa desde Eclipse, lo que significa que fue exitoso el traslado de clases del código entero de *Biogenesis* al proyecto creado en Eclipse para vincularlo con *Max MSP*. El siguiente paso es crear objetos [mxj] en Max y vincularlos a las clases específicas de *Biogenesis* de la manera en que se hizo con la clase *helloWorld* en el tutorial. Estas clases específicas son *Gene.java*, *GeneCode.java*, y según Tyler, *MainWindow.java.*

Se han explorado en su totalidad los códigos contenidos en estas clases y se han encontrado partes específicas de los códigos que manejan las variables que se necesitan para este proyecto. Se tomó nota de dichas partes , pues lo siguiente que hay que hacer es programar dentro de estos mismos códigos para que salgan estas variables por los outlets de los objetos mxj que serán creados.

10 de mayo 2014

Se decidió que es mejor crear clases adicionales que estén vinculadas a las clases existentes de *Biogenesis* en lugar de alterar estas para hacerlas funcionar en Max. Hoy se continuó la búsqueda de asesoría en Java. Se consiguieron dos números y un email. Se está ahora a la espera de unos contactos recomendados por una de las personas que fue contactada, que era a su vez un programador pero que no podía prestar la asesoría.

Se contempló la manera en la que se organizará el *patcher* de *Max MSP,* para que los organismos puedan tener iteraciones polifónicas para un solo color. Los objetos [poly], o [poly~] pueden ser los indicados para esta tarea.

Se averiguó sobre el objeto [mxj]. Para crear versiones dicho objeto se deben programar métodos desde la clase de Java. También en esta clase se determinan los *inlets* y los *outlets,* así como que activa los métodos y que es lo que producen.

La tarea siguiente consiste en elaborar una versión mas avanzada del *patcher* que será utilizado en base a lo que ya se ha venido haciendo en el archivo *ProtoOrganismo.maxpat*. Debe crearse un sistema más completo, que incluya todo lo

referido en el código genético de cada organismo. También es necesario considerar los parámetros de espacialidad.

Al nivel del código, se deben crear clases que se comuniquen con las clases de *Biogenesis* extrayendo de ellas los valores necesitados. Estas deben estar diseñadas para funcionar con *Max MSP* desde *Eclipse*.

11 de mayo 2014

El día de ayer se buscaron más contactos para la asesoría en Java, sin éxito. Se ha estado trabajando directamente en el *patch* de *Max MSP*. Hoy fue posible crear objetos polifónicos funcionales para cada color, así como programarlos de tal manera que puedan recibir en simultanea distintos valores en sus instancias (versiones distintas del mismo *patcher* contenido en el objeto [poly~]). Con esto se ha resuelto en parte la cuestión de la polifonía entre sonidos referentes al mismo color dentro de un solo organismo.

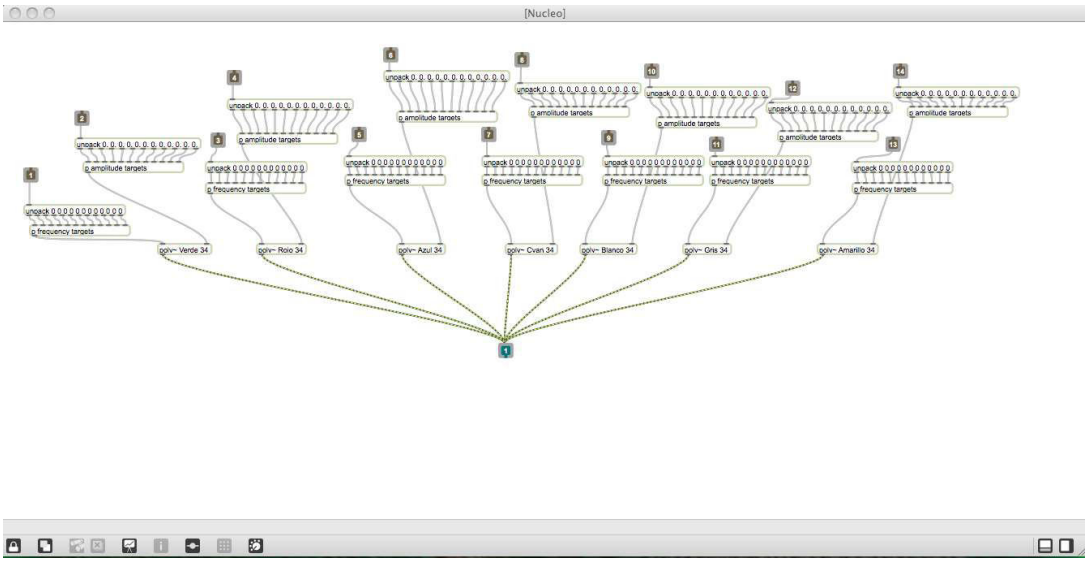


Fig. 24 Sistema que organiza los valores de entrada. (Max MSP)

El problema que actualmente se está intentando resolver es el de cómo dirigir datos numéricos de una lista por una vía u otra del patcher dependiendo del símbolo del color que se encuentra junto a esas variables en la misma lista. De esta manera, las variables en un gen rojo, por ejemplo, tomarán un camino que los lleve al los *inlets* 3 y 4 del *sub patcher* "*Núcleo*" ya que estos corresponden específicamente al color rojo y conducirán a las distintas instancias de un sintetizador polifónico de diente de sierra referente a este color.

13 de mayo 2014

Se ha elaborado un *patcher* complejo en base a las funciones necesarias para la composición generativa al interior de cada organismo. De esta manera se han atacado uno a uno los dilemas técnicos en un proceso gradual de construcción del *patcher*. Muchos de estos "dilemas" son algoritmos de conversión de valores numéricos. Los datos son canalizados por vías específicas dependiendo del color que los acompaña en sus respectivas listas mediante la utilización del objeto [route]. Los datos son luego separados unos de otros por el objeto [unpack] para ser tratados individualmente antes de volver a ser unidos por el objeto [pack]. En este tratamiento se convierten los ángulos de rotación de los genes en las frecuencias de los sonidos producidos por los objetos [poly~] y la longitud de cada gen en la amplitud de su correspondencia sonora, la cual aporta a la idiosincrasia del organismo en medida de la proporción que guarda con las que corresponden a otros genes del mismo organismo.

Los relación de frecuencias, como es determinada por los ángulos de rotación de cada gen es acumulativa, lo que significa que el valor de frecuencia de cada gen se le suma a la frecuencia del gen anterior. Hay un 50% de probabilidad de que el valor de entrada (luego de la primera etapa en la conversión) sea número negativo. Entonces, el la operación siguiente no será una suma sino una resta (la suma de un valor negativo).

Una vez se han convertido estos valores, son empaquetados nuevamente y canalizados en base a su color por una vía que los comunica con los sintetizadores específicos que deben activar.

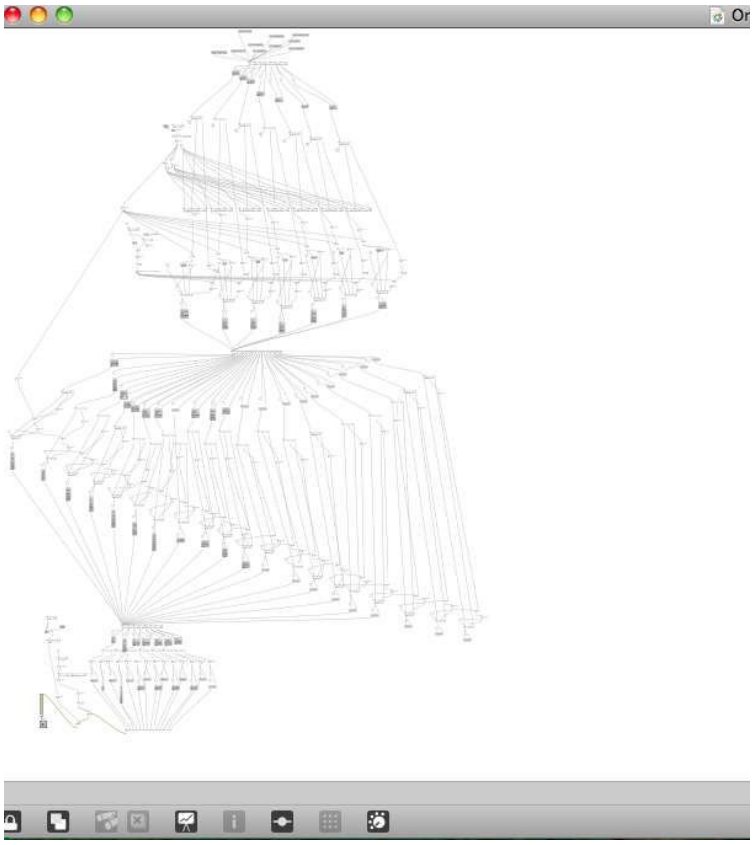


Fig. 25 *Organismo.maxpat* (Max MSP)

Este patcher, aunque funcional, es altamente ineficiente pues por su complejidad excesiva utiliza más del procesador de la que un patcher con sus funciones debería consumir. Está lleno de redundancias. Por ejemplo, el color se canaliza en dos puntos distintos entre los cuales se canalizan los datos en base a su índice. Más eficiente sería canalizar los datos por su índice desde el principio y más adelante por su color.

Aunque sea innecesariamente complejo y lleno de redundancias, la construcción de este patcher fue un paso clave en la conceptualización de cada proceso, pues en el se resolvieron paso a paso los aspectos de conversión de los datos como resultan necesarios para el proyecto, y también se vincularon estos entre sí mediante algoritmos que surgieron como solución a cada uno de estos problemas. Se construirá eventualmente una versión de este *patcher* mucho más parsimoniosa y eficaz.

14 de mayo 2014

Se ha creado una versión del *patcher* para el organismo mucho más limpia, simplificada y eficiente que la que se creó el día de ayer. Cumple las mismas funciones, pero lo hace de una manera mucho más ordenada. El resultado es un patcher que no utiliza tanto CPU como el anterior. La implementación del *subpatcher* "*Núcleo*" se abandonó.

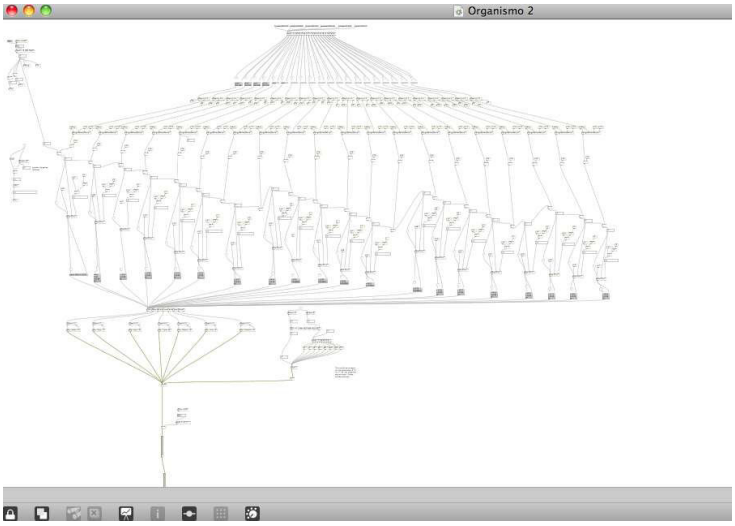


Fig. 26 *Organismo 2.maxpat* (Max MSP)

Este empieza canalizando las listas con los datos, cada una de las cuales vendría representando a un gen, por una vía determinada por su número índice el cual también determinará la relación en términos de frecuencia con los otros genes en el mismo organismo.

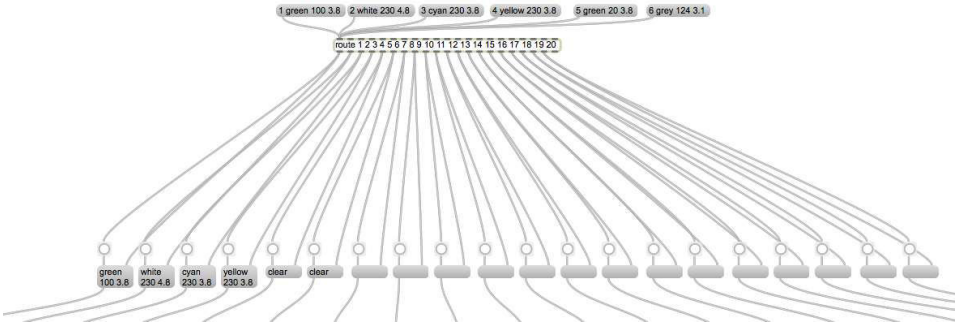


Fig. 27 Sistema organizador de datos por color en *Organismo 2.maxpat* (Max MSP)

Estos datos son separador por el objeto [unpack] para ser tratados por caminos específicos. La frecuencia es determinada en base a los valores del ángulo de cada gen en función de un sistema que determina si el ángulo ocurre en una dirección o la otra. En este sistema, 270 grados se vendrían a tratar como -90 grados, entendiendo una frecuencia base como el eje. Los límites de esta frecuencia están determinados por una

octava superior y una inferior a la frecuencia base. El dato de la frecuencia base es arrojado en flujo continuo por una onda sinusoidal de baja frecuencia cuya fase es extrapolada a datos numéricos. El número del ángulo contenido en el gen es convertido a valores de escala congruente mediante el objeto [scale] para colocarse en una posición proporcional a la frecuencia base. Cada gen genera una proporción diferente, la cual se mantiene incluso mediante el flujo de datos continuo y gradual de la frecuencia base. El resultado es un acorde poli tímbrico con portamento.

Dependiendo del orden de los genes, el cual está indicado por el índice al principio de la lista en el comienzo del *patcher*, estas frecuencias resultantes se suman unas a otras empezando por la suma de la frecuencia del primer gen, partiendo de la frecuencia base de todo el organismo de la cual se extrajeron también los límites del proceso de escalamiento que se describe en el párrafo anterior. Hay casos en que el numero que se suma es negativo, por lo que no se puede decir que haya necesariamente una acumulación de valores ascendente.

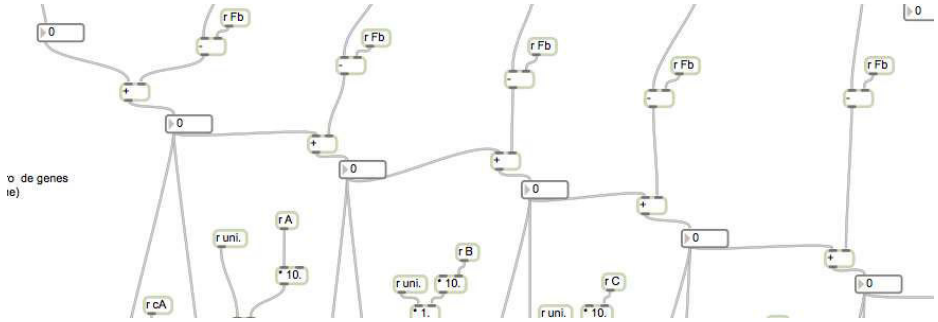


Fig. 28 *Organismo 2.maxpat* (Max MSP)

Por su parte, un sistema procesa los valores de longitud del gen para convertirlos en la amplitud que le corresponderá a cada gen. Este sistema divide uno por el número de genes, siendo el número uno representativo de la amplitud total disponible para ese

organismo en particular. Este número se divide por 180, que es el número de valores disponibles en el contexto de la longitud de los genes donde la longitud máxima de cada segmento es 18.0. Con esto se genera una unidad de amplitud que es relativa al número de genes del organismo. Esta unidad se manda a cada uno de los canales (para cada gen) donde es multiplicada por la longitud del gen multiplicada por diez. Con esto se genera un valor para la ganancia de ese gen que no excederá al número uno, ni siquiera cuando sea sumado a los mismos valores de ganancia de los otros genes.

La información de la amplitud y la frecuencia del gen es empaquetada luego junto al color del gen para generar una lista de valores procesados por el algoritmo de conversión. El color del gen determina en esta lista la vía por la que estos datos serán canalizados para que así se puedan introducir en el sintetizador correcto, es decir, aquel que es representativo para el color de ese gen.

Los valores de simetría y espejo que están contenidos en el código genético determinan una modulación a la amplitud de todo el organismo. Cuando el organismo no está en modo espejo, el valor de simetría viene a representar la frecuencia de la onda moduladora que es suficientemente baja para producir un efecto perceptual de iteraciones rítmicas regulares. En este esquema, una simetría de uno produciría un sonido homogéneo. Una simetría de dos produciría una iteración por segundo, una de tres produciría dos iteraciones por segundo, y así sucesivamente hasta una simetría de ocho que produciría siete iteraciones en un segundo. Cuando la simetría es de tipo espejo, estos valores de simetría son direccionados a canales por los que son remplazados por una serie de valores Fibonacci a partir del número 8.

27 de junio 2014

Se han retomado las labores asociadas a la construcción del proyecto. Se ha contactado por vía de Ana María Romano al compositor Daniel Zea, con quien actualmente se mantiene diálogo por vía de emails. Es posible que se haga con él una reunión vía *Skype* para discutir los detalles técnicos del proyecto. Recomendó unos cursos que el ha grabado que están disponibles en su página, ante todo para tratar el tema de los objetos [poly~] (objetos que permiten distintas iteraciones de sonidos o sistemas, para en el primer caso generar polifonía y en el segundo economizar procesos), que son necesarios para la composición. Con el propósito en mente de simplificar procesos para que los algoritmos no sobrecarguen la capacidad del CPU, así como para hacer mas eficientes y parsimoniosos los procesos, se ha contemplado la utilización de objetos [poly~] para operaciones de MSP que van más allá de la generación de tonos. Los canales por los que viajan los paquetes específicos de datos pueden estar contenidos dentro de objetos [poly~] para que las iteraciones se vayan creando en la medida en que se necesitan, siendo estos canales, dentro del organismo, por donde fluye la información pertinente a cada gen. En el enfoque del mundo, el cual contiene muchos organismos distintos, cada objeto [poly~] contendrá entonces un organismo. Esto implica la presencia de objetos [poly~] dentro de otros objetos [poly~], con lo que surge una dinámica de jerarquías sistémicas, es decir, un sistema multinivel que involucra sistemas dentro de sistemas.

Adicionalmente, se ha realizado un tutorial de Java (https://www.youtube.com/watch?v=3u1fu6f8Hto) donde se han estudiado los conceptos básicos que involucra este lenguaje de programación multiplataforma de manera más exitosa que en los intentos previos. Se aprendió sobre los *métodos*, los

datos, las *clases* y los *objetos*, y también como especificarlos mediante la escritura del código.

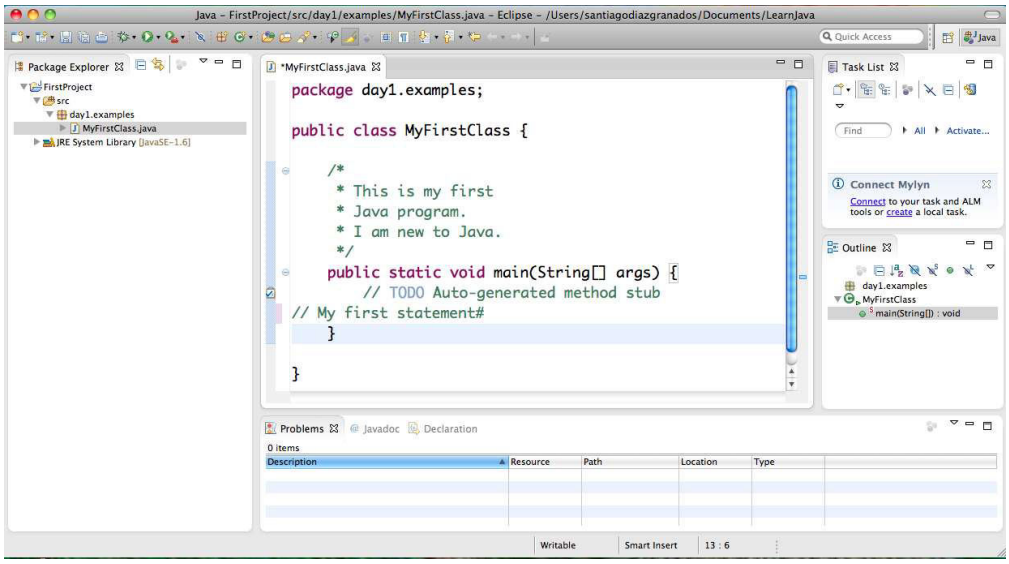


Fig. 29 Tutorial de Java (Patrick Washington, 2012, https://www.youtube.com/watch?v=3u1fu6f8Hto)

1 de julio 2014

Por la buena calidad del tutorial de Java, se optó por descargarlo todo. Consta de un video de cuatro horas de duración donde se detallan los protocolos paso por paso. Hoy se realizó aproximadamente media hora adicional con lo que se completó hasta la mitad de la totalidad del video. Se aprendió como mandar datos desde las clases principales hasta las que están representadas por los objetos. Se conoció el comando *return,* que a diferencia de *void*, devuelve datos a los métodos para que a parte de ser impresos puedan ser utilizados en las operaciones matemáticas. Se exploró el concepto de *polimorfismo*, con el que se asignan datos a métodos específicos dependiendo del tipo de datos que son. Se aprendió también como obtener datos privados mediante *setters* y *getters.*

Ya se confirmó por medio de los foros de *Max MSP*, que sí es posible colocar objetos [poly~] dentro de objetos [poly~].

Se ha decidido adicionalmente que para quitarle peso al CPU, se realizarán muchas de las operaciones matemáticas no en Max, sino directamente en el proyecto de Java. Serán programadas desde *Eclipse*.

Queda pendiente la reunión vía Skype con Daniel Zea.

7 de julio 2014

El sábado pasado se hizo la reunión vía Skype con Daniel Zea. Encontrándose él en Ginebra, Suiza, enseñó como crear las distintas instancias para un objeto polifónico. Con este aprendizaje se procedió a la generación de modelos para esquematizar la serie de procesos involucrados en la composición. El día de hoy se creó una tercera versión del organismo que en el patch padre que se denominará *mundo.* Será un *sub patcher*. La versión nueva asume que se relegarán procesos algorítmicos a la parte de la composición que estará programada en java para aliviarle así la carga a la CPU. En ese sentido es mucho más simple y limpia que la versión segunda.

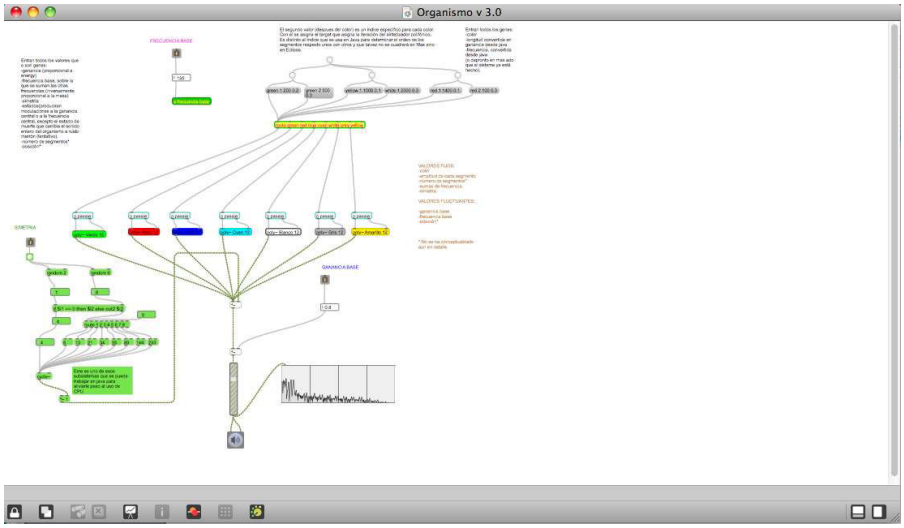
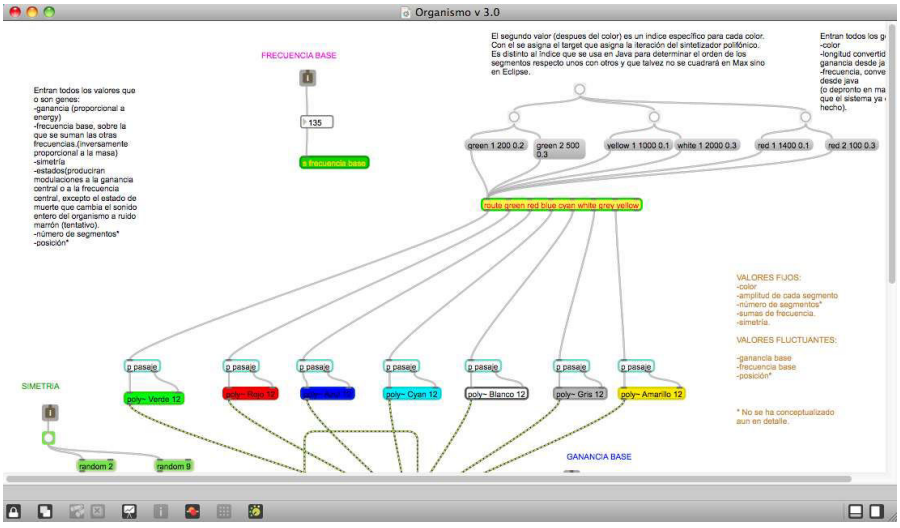
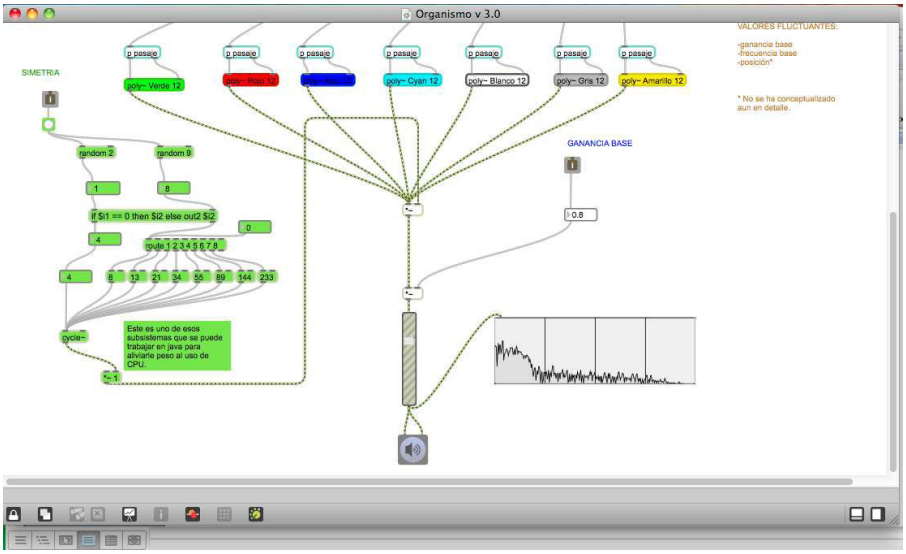


Fig. 30 Organismo v3.0.maxpat (Max MSP)

La preocupación metodológica principal, que era la generación de eventos polifónicos de un mismo tipo de onda, se logró satisfactoriamente luego de una extensa labor analítica y desarrollativa.





Figs. 31 y 32 Funciones del patcher (Max MSP)

Ya teniendo más claro el desarrollo estructural al nivel de Max, es prioridad ahora dedicar más tiempo a aprender a programar en Java para programar las clases de java que tomarán valores de las clases de *Biogenesis*. Las siguientes estarán entonces enfocadas en esa dirección.

9 de julio de 2014

Se completó el tutorial de Java dictado por Patrick Washington. Se logró hacer correr *Biogenesis* desde una clase externa al paquete de *Biogenesis* mediante la importación de clases de *Biogenesis* a la clase creada. Hay dificultades en la creación de objetos particulares. Se le ha pedido ayuda a Tyler del foro de Biogenesis y se está esperando una

12 de Julio 2014

Se han explorado un nivel de profundidad mayor del código del programa, principalmente en lo pertinente a organismos individuales. Los parámetros del mundo no pueden ser alterados cuando el programa corre desde *main.class*. Si acaso es deseable alterarlos (probablemente si va a serlo), tocará hacerlo desde el código en java. Aunque ya se ha logrado imprimir datos desde Eclipse a Max, el método que hace esto toca colocarlo por fuera del método principal, pues si se coloca adentro no funciona. Una solución podría ser mandar estos datos desde clases alternas.

14 de Julio 2014

Es factible utilizar métodos matemáticos en las declaraciones de las variables para traducir desde Java las variables como son necesitadas en Max.

15 de Julio 2014

Los últimos días han sido de trabajo arduo pero de pocos avances. La dificultad actual consiste en encontrar la manera de hacer las programaciones necesarias, pero dada la falta de experiencia en programación que se posee, esto ha un reto. Se está atacando el problema por todos los flancos: pidiendo ayuda en foros, comunicándose con Tyler, y viendo tutoriales de Java tanto escritos como en video.

16 de Julio 2014

Las tareas que hay que realizar en el medio de Java son las siguientes:

-Imprimir un flujo de variables continuas en la consola.

-Imprimir a Max desde la clase que se está elaborando.

-Crear métodos con las operaciones matemáticas necesarias para su traducción a variables musicales coherentes.

Adicionalmente, hay que diseñar el mundo de los organismos en Java, y decidir si se usarán *subpatchers* o objetos [poly~] para la reproducción de varios organismos al tiempo.

18 de Julio 2014

Se realizó una conferencia en línea con el ingeniero Andrés Uribe Gonzales, en la que dilucidó la manera en la que se deben llevar a cabo las operaciones a realizarse en Java. Con su colaboración se logró satisfactoriamente imprimir en la consola de Max una serie fluida de valores fluctuantes referentes en este caso al nivel de oxigeno en el mundo virtual. Para esto se utilizó en Max un objeto [metro] que manda *bangs* consecutivos al objeto [mxj Oxigeno] relacionado con la variable *oxigen*.

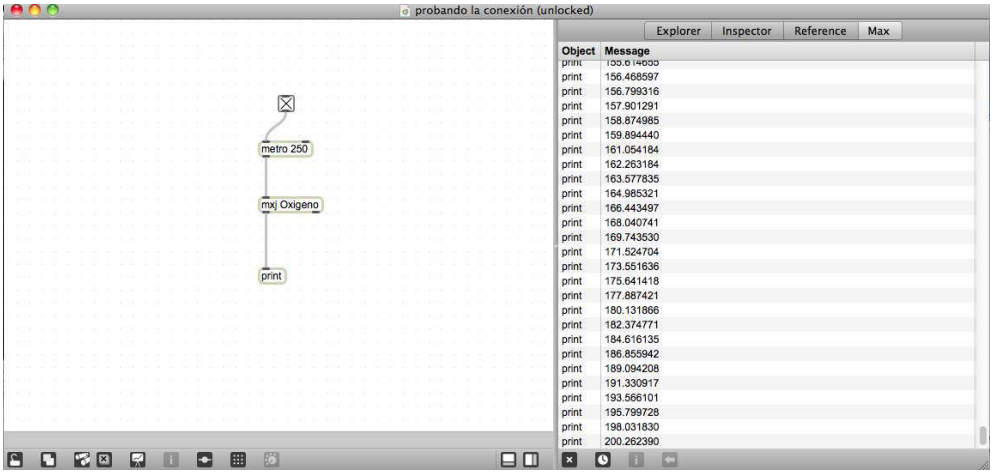


Fig. 33 Max imprimiendo variables de *Biogenesis*. (Max MSP)

Para evitar que se cree una ventana nueva de *Biogenesis* con cada *bang,* lo que resultaría en el colapso de todo el proceso, se creó una clase de tipo *ClassicSingleton.*

Esta permite referirse a la *MainWindow* al mismo tiempo que evita que aparezcan muchas versiones de ella.

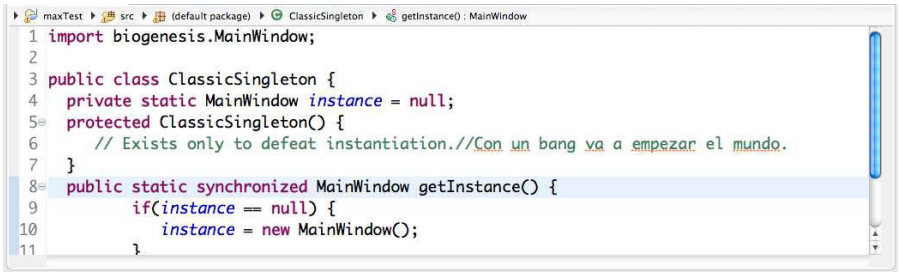


Fig. 34 *ClassicSingleton.java* (Eclipse)

11 de agosto 2014

Persiste un problema: no se sabe como crear *outputs* en las clases de Java referentes a objetos [mxj] en Max. Este constituye un bloqueo en el proceso, y se estima que cuando se resuelva esta incógnita particular, se va a poder seguir avanzando con la creación del sistema.

Una observación: se pueden adicionar a los datos que pasan de *Biogenesis* a Max los índices referentes a genes y a organismos particulares, para que se dirija en forma ordenada la información pertinente por el camino correcto.

12 de agosto de 2014

Se consiguió cambiar los parámetros predeterminados del mundo en la clase del paquete de *Biogenesis* que se llama *Utils.java*. Todas las modificaciones a los parámetros del mundo se deben hacer acá, pues por alguna razón no se pueden modificar en la ventana del programa.

Con la ayuda de los usuarios del foro de *Max MSP*, pudieron crearse *outlets* varios en las clases. Resulta que había que incluir las declaraciones de los *outlets* en el constructor de cada clase. Aún no se había creado dicho constructor, así que se creó.

16 de agosto 2014

Se sigue avanzando en la construcción del sistema. El énfasis ahora esta en Max donde tiene que construirse un *patcher* adecuado a las necesidades de la composición. Ya se logró conectar la información proveniente de *Biogenesis* al organismo. Este está contenido en un *subpatcher* de *Mundo.maxpat*. El problema es que este *subpatcher* está recibiendo información de todos los organismos al tiempo, por lo que ese necesario crear un sistema que ordene los valores de acuerdo a los índices que son los argumentos primeros en las listas que los representan.

No se sabe aún cual es el objeto indicado para este propósito pero se cree que puede ser parecido al objeto [route], que se ha utilizado ya varias veces. A diferencia de [route], dicho objeto debe ordenar una cantidad indefinida de valores que llegan a él desde Biogenesis. Queda pendiente averiguar cual objeto funciona para esto, y en caso dado, pedir ayuda nuevamente en los foros de Max MSP.

19 de agosto 2014

Hoy se realizó una reunión con el compositor Daniel Zea, ya no por Skyppe, sino en persona. La reunión se enfocó en la resolución de cuestiones particulares a la construcción del *patcher*. Se resolvió como crear un sistema cuadrafónico que produzca la sensación de los organismos en el espacio. Se revisó la manera en la que se estaban manteniendo los intervalos particulares de cada organismo proporcionales a una

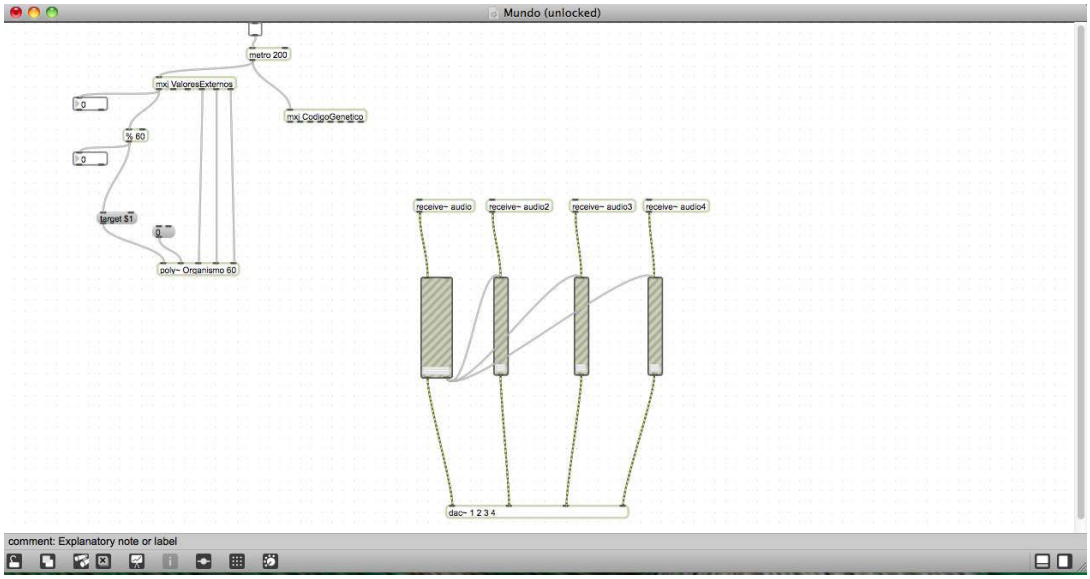
frecuencia que demuestra fluctuaciones paulatinas. Daniel elaboró una forma más eficiente de llevar a cabo esta tarea que la que se venía utilizando. Se introdujo el objeto [modulo], el cual permite crear índices limitados a partir de números entrantes ilimitados que representan a cada organismo que nace. Con el objeto modulo se resolvió una de las dificultades principales hasta el momento: la de ordenar los organismos en Max. Se optó finalmente por utilizar un objeto [poly~] cuyas iteraciones representan a cada organismo, en oposición a crear muchos *subpatchers* latentes aguardando a ser ocupados por información referente a organismos que van naciendo. Con esto además, se consigue que el *patcher* se vea más limpio. Lo que queda entonces es un sistema de objetos polifónicos dentro de objetos polifónicos.

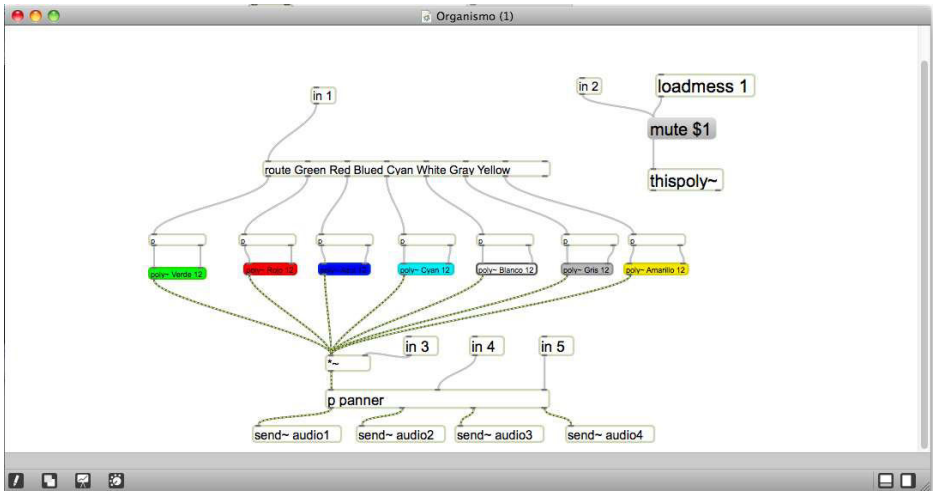
Se crearon muchos de los sistemas del patcher en la reunión misma. Con lo que se desarrolló en este encuentro se sentaron las bases para la creación del *patcher* de la composición generativa y ya está mas clara la manera en la que se completará el proyecto.

Daniel alentó a que se prosiga cuanto antes para poder trabajar directamente sobre el sonido, para de esta manera, no descuidar el aspecto artístico musical de la composición.

20 de agosto 2014

En base a lo que se desarrolló junto a Daniel Zea, se ha creado una nueva versión del mundo, así como una nueva versión del organismo polifónico.





Figs. 35 y 36 Dos niveles del sistema: *Mundo* y *Organismo* (Max MSP).

27 de agosto 2014

Se siguió trabajando hoy en la construcción del patcher. Ante un problema con los objetos [sig~], que no funcionan por alguna razón dentro de [poly~ Organismo], se intentó en lugar de utilizar un objeto poly para los organismos, usar mas bien *subpatchers*. De esta forma existirán los *subpatchers* varios en estados latentes hasta que sean activados por las variables referentes a los organismos. Por ahora se pretende limitar la cantidad de organismos musicalizados a treinta, y en caso de que halla en el

mundo más de treinta, simplemente se empezarán a musicalizar los nuevos al haberse rotado a través de todos los *subpatchers* disponibles mediante la utilización del objeto módulo ([%]). El problema ahora es que por alguna razón, los sintetizadores específicos de cada color están perdiendo todos una entrada, y la única salida que tienen. No se sabe el porque de esto.

Daniel Zea está intentando comprender que es lo que esta haciendo mal el patcher por su parte y se espera aún respuesta de él. Así mismo, se está esperando una repuesta de Tyler Coleman con respecto a un problema en Java; el índice del gen no es un valor del tipo necesario para poder ser enviado por el *outlet* en el objeto [mxj].

28 de agosto 2014

Se corrigió por fin el error en Java con la ayuda de Tyler. Con eso resuelto, es posible diseñar un sistema para el ordenamiento de datos genéticos en Max. Mediante un proceso largo de prueba y error, parecen haberse logrado avances. Se desechó, por el momento, la idea de los *subpatchers*, y se retomó el método de los [poly~]s . Se ha logrado producir un sistema sonoro que monitorea el nacimiento de los organismos así como su energía y su masa. Un problema actual es que el sistema polifónico no está funcionando, pues unos sonidos enmascaran a otros completamente, solo siendo evidente la presencia de muchos sonidos en una fluctuación que se da entre ellos en forma desordenada.

4 de septiembre 2014

Se ha decidido nuevamente optar por *subpatchers* en lugar de objetos polifónicos. Es más fácil organizar los datos en esta forma. Mediante la construcción de un nuevo

patcher en esta forma, se descubrió que muchas de las dificultades del *patcher* previo, en particular en cuanto al solapamiento erróneo de las señales, se debían a que los objetos *send* y *receive* no estaban siendo apropiadamente referenciados para cada instancia de los objetos polifónicos. De pronto es posible hacer esto, pero en principio parece demasiado complicado. Se está desarrollando el nuevo *patcher* y los prospectos parecen prometedores.

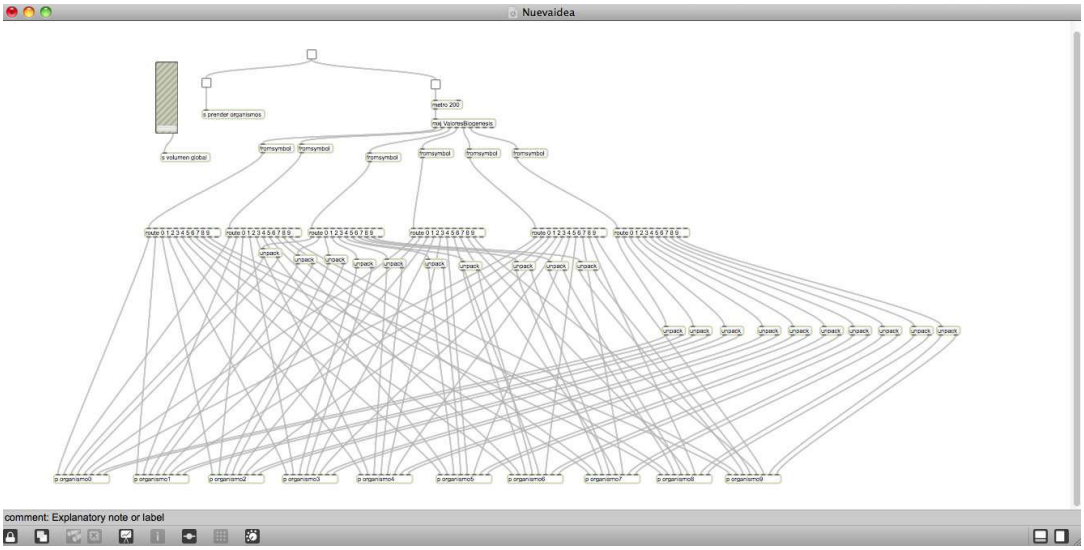
12 de septiembre 2014

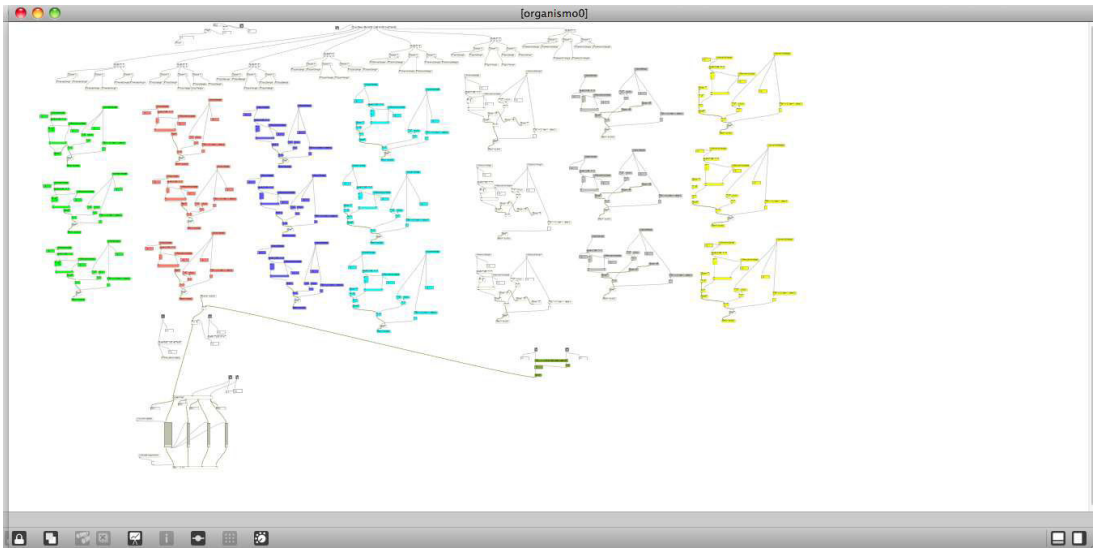
Se ha cumplido la meta principal. Se tiene una pieza generativa evolutiva que representa las características fenotípicas de los organismos que componen un ecosistema virtual. Se limitó el número de genes a tres. El número de organismos que capta el programa es 10. Después de esto, los organismos ya no son procesados para ser convertidos en sonidos.

Por lo tanto, aunque ya se tiene una composición, hay que mejorarla. En primer lugar, hay que implementar un sistema para que los organismos siendo convertidos en sonido se roten los espacios disponibles en los sintetizadores a medida que aparecen para que así la composición generativa pueda seguir. Además, hay que programar una función para que los organismos se apaguen cuando se mueran, lo que actualmente no está ocurriendo. También hay crear sistemas que respondan a las acciones de los organismos (atacar, ser atacado, etc.) para que estas sean convertidas en sonido. Otra cosa que falta por hacer, que no es necesaria pero vale la pena intentar implementar puesto que estaba entre las intenciones iniciales, es un sistema que convierta los cadáveres de los organismos en ruido, es decir, que detecte cuando un organismo muere y deja el cadáver para producir un ruido de algún tipo estándar como representación del

mismo, que igual dependerá de valores de energía para su ganancia y valores indicadores de su posición en el espacio.

Hay también que considerar cual será el título de esta composición.





Figs. 37 y 38 El nuevo *patcher*, nivel del mundo (Fig. 72), nivel del organismo (Fig. 73). No se implementan objetos polifónicos. (Max MSP)

16 de Octubre 2014 (entrada posterior a la primera entrega del trabajo)

Se logró que la obra sonara en forma indefinida. Se implementó exitosamente un sistema para silenciar a los organismos muertos.

Se pasó todo el proyecto a un computador *iMac* puesto que la interfaz de audio que se tiene no se puede conectar al portátil con el que se ha venido trabajando hasta el momento. La interfaz posee varias salidas por lo que construir la cuadrafonía no va a ser un problema. Es necesario ahora afrontar los dilemas logísticos de la construcción de la instalación sonora, como hablar con el decano de la facultad de artes y pedir un *videobeam* prestado.