Una caracterización a la resolución de problemas colaborativa distribuida desde los sistemas multiagente

Offray Vladimir Luna Cárdenas

Jaime Parra Bernal
Director de Trabajo de Grado

Pontificia Universidad Javeriana
Facultad de Educación
Maestría en Educación
Línea de Cognición, Creatividad y Aprendizaje

Bogotá D.C.
Junio de 2007

Resolución colectiva de problemas en sistemas multiagente

Una caracterización a la resolución de problemas colaborativa distribuida desde los sistemas multiagente

Offray Vladimir Luna Cárdenas

Aprobado

Jaime Parra Bernal
Director de Trabajo de Grado

Jurado 1

Jurado 2

Bogotá D.C.
Junio de 2007

Resolución colectiva de problemas en sistemas multiagente

Resumen

El presente trabajo investigativo tiene como objeto caracterizar la resolución colectiva de problemas en sistemas multiagente. Desde el abordaje teórico se asumió una postura considerando la escuela socio histórica rusa, la cognición distribuída y la teoría de sistemas multiagentes y en cuanto a lo metodológico se optó por el modelaje y la simulación como una forma de aproximarse a la compresión y a la caracterización, al mismo tiempo que se establecía un correlato de aula que dialogara con el modelo para que mutamente se inspiraran y cambiaran. Se llegó a un modelo que caracteriza la resolución colectiva de problemas en sistemas multiagente a través de dos parámetros de los agentes individuales: coherencia y compentencia y uno del problema: modularidad y se valido el modelo a través de evidenciar fenómenos emergentes tanto en el modelo como en su correlato de aula.

Palabras Clave: cognición distribuida, sistemas multiagente, simulación, escuela sociohistórica, Kedama, Squeak, Bots Inc, diseño de entorno, educación.

Advertencia

La Universidad no se hace responsable por los conceptos emitidos por los alumnos en sus trabajos de tesis, sólo velará por que no se publique nada contario al dogma y a la moral católica y por que las tesis no contengan ataques personales contra persona alguna, antes bien se vea en ellas el anhelo de buscar verdad y justicia.

Agradecimientos

Durante estos tres años muchas personas tuvieron que ver particularmente con el proceso que me ayudo a terminar este trabajo. Quisiera mencionar y agradecer especialmente

A mi papá por escuchar y recordarme las cosas verdaderamente importantes.

A mi mamá por su permanente e infinito apoyo e impulso.

A mi hermanos por los valiosos aportes desde la diferencias.

A Jaime Parra por la inspiración inicial y la de los posteriores *momentos "aja!"* y Fabio Enrique Martínez por la excelente panorámica sobre sistemas multiagentes.

A Andrea, Gloria y Laura, *las ángeles de Charlie*, por las cuotas de cielo.

A Luis Alejandro Bernal, Fredy Guio y Carlos Manuel Estevez-Bretón por la buena tertulia y la literatura.

A los amigos del rol, por ayudarme a mantenerme no tan cuerdo.

A *Maka*, por recordarme la entrada al laberinto y ayudarme a recorrerlo.

Dedicatoria

A mi familia, en particular los que empiezan a hacer parte de ella, como Jonathan y Nicolas

Índice

Introducción

La escuela socio histórica rusa y la de la cognición distribuída han puesto de manifiesto la importancia del entorno, las mediaciones, los otros y lo sociocultural en el desarrollo cognitivo y también el hecho de que la inteligencia no sólo están "dentro" de la cabeza, sino que ella emerge y se da en y gracias a lo colectivo. El auge del software social, en sitios como la Wikipedia, SourceForge y sus proyectos de software libre, Amazon, Youtube, muestran explicitaciones informáticas de estas inteligencias colectivas. Cómo se da y opera esta inteligencia, cómo se relacióna con lo individual, cómo se establecen "continuos" y otras tantas inquietudes son parte de este campo relativamente novel, con importantes desafíos teóricos y abrumadoras consecuencias prácticas, en particular y desde el tema que nos atañe, en lo cognitivo y lo educativo. Esta tesis pretende ser una aproximación al estudio de esta inteligencia colectiva y sus *continuos*, específicamente indagando por la resolución colectiva de problemas.

Para dicha indagación se han tomado dos caminos dialogantes: un modelo algorítmico con implementación computacional y un correlato de aula. El último se ha abordado con un enfoque que en el se fundamente en la escuela postvigostkiana y de la cognición distribuída y en experiencias de trabajo con estudiantes universitarios durante tres semestres usando mediaciones tecnológicas de software social (Wikis) y entornos de programación orientado a objetos, multimediales y de programación gráfica (*eToys*) o con un fuerte elemento visual (*Bots Inc*). El primero a asumido metodológicamente la aproximación desde la modelación sintética o basada en multiagentes, del que algunos autores (Pfeifer y Scheier, 1999 y Macal y North, 2005) mencionan ser un tercer enfoque entre lo analítico y lo sintético o entre lo inductivo y lo deductivo. Este lugar se hibrida aún más si se consideran las interacciones entre el modelo formal y la experiencia de aula y fue el escenario fascinante donde se dió esta indagación, entre (des)encuentros, reconcialiciones y *momentos "aja!"*, fruto de la inspiración, pero sobre todo y como dice el refrán, de la transpiración.

Para realizar con el lector un viaje compartido, en primer lugar se ha dispuesto un lugar de antecedentes, planteamiento del problema y los objetivos; en segundo, se ha hecho un revisión teórica de los conceptos de multiagente a la luz de lo que nos atañe en cuanto a resolución de problemas; en tercero se ha desarrollado un modelo formal de resolución de problemas, que se cree aporta elementos nóveles al involucrar aspectos que en la

literatura, hasta donde se pudo consultar, no se encontraban en un único modelo; en cuarto lugar se ha mostrado el correlato de aula del modelo (que está complementado en los anexos con capturas de pantalla del trabajo de los estudiantes así como muestras de las huellas de sus interacciones en el software social) y finalmente se han establecido las conclusiones y recomendaciones.

Creo, al igual que Susan Josephson en *Thinking like a machine*, que respecto a los modelos uno pueden considerar a los mismos como descripciones o adscripciones, dependiendo de cómo sirvan al propósito de la comprensión. Cuando estamos dentro de las descripciones, pensamos que el modelo es más "fiel a la realidad", mientras que en las adscripciones nuestros modelos hablan precedidos por un "*es como si...*". La ciencia cognitiva, la cognición distribuída y los sistemas multiagente son enfoques muy recientes y en ese sentido es labor del lector en qué lugar entre adscripción y descripción coloca al modelo acá presentado.

El autor espera que el lector encuentre en este viaje uno o varios lugares de su agrado, bien sea en lo formal, lo computacional, lo cognitivo o lo educativo para que podamos pensar y/o asumir, en un tiempo no muy lejano, algún problema en conjunto y colectivamente.

1 Antecedentes

La cognición distribuida dentro de la ciencia cognitiva, es un enfoque relativamente novel que data de mediados de los ochentas y fue nombrado así después de los aportes de Ed Hutchings y sus colegas de la Universidad de California, San Diego (Hutchins, 1995), si bien los orígenes pueden rastrearse hasta la escuela histórico-cultural rusa, con Vigostsky, Luria y Leontjev como sus principales exponentes y quienes hicieron los primeros aportes a esta mirada, de acuerdo con Rizzo y Marti, (sin fecha). En virtud de que esta tesis intenta explorar la relación existente entre la triada agentes, resolución de problemas y comunidades, se cree que la aproximación desde la cognición distribuida aporta un marco conceptual, teoríco y metodológico para mirar esta relación más conveniente que el podrían aportar por separado, la cognición corpórea (Pfeifer y Scheier, 1999), las redes de aprendizaje, (Teles, Harasim y otros, 2000), el de las nuevas mediaciones Web en línea (Kerckchove, 1999).

Decortis, Noirfalise y Saudeli (sin año) mencionan que la cognición distribuida tiene dos marcos de ocupación: "Por una parte, su objetivo es estudiar las representaciones del conocimiento internas y externas al individuo. Por otra parte, está interesada en la propagación del conocimiento entre los individuos, y los artefactos, y las transformaciones sostenidas por las estructuras cuando son usadas por individuos y artefactos. Esta nueva aproximación permite el estudio del fenómeno cognitivo no observable al nivel individual, tal como el trabajo coperativo en una tarea socialmente distribuida".

Existen proyectos que intentan el uso de sistemas de software para modelar el conocimiento y se basan en los aportes comunitarios a dicho sistemas, como es el caso Mindpixel[1], Openmind[2], OpenCyC[3], ConceptNet[4] y WorldNet[5], sin embargo los dos primeros aún no han liberado productos de software que sean fácilmente reutilizables o modificables para otros contextos. El último tiene un motor semántico con un lenguaje declarativo que almacena alrededor de un millón seiscientos mil acertos sobre el mundo. El caracter abierto del mismo ha permitido explorar la posibilidad de enlazar el motor semántico con

1. http://www.mindpixel.com/chris/
2. www.openmind.org y también http://openmind.media.mit.edu/
3. http://www.opencyc.org/
4. http://web.media.mit.edu/~hugo/conceptnet/
5. http://wordnet.princeton.edu/

agentes declarativos, como los robots de chat, tal como lo muestra el proyecto AIMLpad[6]. Otro sistema exitoso en el uso comunidades y artefactos cognitivos que explicitan como representaciones externas del conocimiento algunas representaciones internas de sus miembros ha sido el proyecto Wikipedia[7], que a la fecha ha alcanzado un número de un millón de artículos enciclopédicos escritos en más de 100 lenguas, en tan sólo 3 años.

Dado sus objetos de interés, la cognición distribuida ha sido empleada para el estudio de la organización dinámica de los Sistemas Multiagente o MAS (Flor, 1994) y en los equipos de programadores y programación entre pares (Williams, Kessier, Cunningham, 2000), el diseño de ambientes interactivos de aprendizaje (Dillenbourg, 1995), el estudio de casos en equipos de navegación de naves marítimas o aéreas (Hutchings, 1995), entre otros.

Una de las propuestas más interesantes de estudio fue hecha por un equipo interdisciplinar en su *Research Proposal for a "Geconcerteerde Onderzoeksactie"* y se presentó en el escrito *Modelling the Emergence and Evolution of Distributed Cognition* (Heylegen *et al*, 2004), que critica la aproximación de Hutchins al estudio de la cognición distribuida, diciendo que "este estudio identifica las diferencias computacionales del sistema como un todo versus aquellos de una actividad de tarea individual", pero inmediatamente establece que "a pesar de sus promesas, la aproximación de la cognición distribuida aún ofrece poco más que una colección heterogénea de ideas, técnicas de observación, simulaciones preeliminares y casos de estudio. Carece de un marco de trabajo teórico coherente que podría integrar los varios conceptos y observaciones, y proveer un fundamento sólido para la construcción de sistemas y procesos concretos. [...] La presente propuesta intenta desarrollar tal teoría integrada, soportada por observaciones, experimentos y detalladas simulaciones de computador." A partir de su postura crítica plantean cinco hipótesis de trabajo y un proyecto a 5 años, empezando en el 2005, que en caso de ser financiado, con un costo de 743310 euros, podría ayudar en la configuración, exploración y validación de las hipótesis. Estas hipótesis son (pág 1).:

- Los grupos de agentes se auto-organizan para formar un sistema diferenciado, coordinado, adaptado a su ambiente.

6. http://209.168.21.76/AIMLpad/

7. www.wikipedia.org

- El sistema opta cooperativamente por medios externos de para la propagación de la información.

- El sistema cognitivo distribuido resultante puede ser modelado como una red conexionista que aprende.

- La información en la red es propagada selectivamente a través de los siguientes criterios: utilidad, novedad, coherencia, simplicidad, formalidad, expresividad, autoridad, conformidad o consenso.

- El conocimiento novel emerge a través de interacciones recurrentes no lineales.

Los agentes, bien sean naturales o artificiales, han sido parte de los estudios anteriores, pero el estudio de caso de comunidades virtuales donde intervengan agentes naturales y artificiales, hasta donde la consulta bibliográfica lo permitió, aún no ha sido emprendido o no cuenta con material ampliamente publicado que haya sido encontrado. Es en particular el item tercero de esta lista el que permite intentar explorar, desde un modelo multiagente, los cuatro items restantes en comunidades mediadas por NTICs, en particular el asunto central de esta tesis, la resolución colectiva de problemas.

2 Problema

Caracterizar cómo se resuelven problemas en colectividades a traves de sistemas multiagente

3 Justificación

Los autores de *Modelling the Emergence and Evolution of Distributed Cognition*, nos ofrecen al mismo tiempo preguntas cuyas respuestas puedan provenir de un estudio como este aplicaciones potenciales de los resultados, sin embargo no ofrecen una reflexión en el correlato educativo, cosa que se presentará después de revisar la posición de esos autores.

El autor considera que, si bien en un marco más modesto, se pueden explorar importantes preguntas similares a las planteadas por el estudio, como son:

- Cómo agentes inicialmente independientes a través de la interacción (usando medios externos) llegan a formar un sistema cognitivo distribuido?
- Qué clase de coordinación entre las diferentes actividades de procesamiento de información emerge?
- Cuál es conocimiento nuevo o emergente en este sistema, es decir, conocimiento que antes no existía en la mente de los agentes individuales?
- En qué medida esta cognición emergente es mejor o peor que la cognición inicial individual?
- Qué características influencian la eficiencia del proceso? Por ejemplo, en qué medida las capacidades cognitivas resultantes dependen del número de agentes, la diversidad de la experiencia entre agentes, o la presencia o ausencia de diferentes tipos de medios?

En cuanto a las aplicaciones Heylen y otros. (2004, pág 5) consideran:

- La comprensión de cómo el conocimiento y la información son distribuidas a través de sistemas sociales nos ayudaría a impulsar el desarrollo económico y social que el nuevo conocimiento y la mejor coordinación engendran. [Martens, 1998; 2004]. Tal teoría nos diría cómo las ideas nuevas importanes pueden difundirse más eficientemente, y recíprocamente como el esparcimiento de rumores falsos, supersticiones y "parásitos de información" pueden ser cortado [Heylighen, 1999]. Más generalmente, puede ayudarnos a controlar los prejuicios cognitivos y sociales cuya ubicuidad los sicólogos han ampliamente demostrado [Brauer et al., 2001; Klein et al., 2003; Van Rooy, Van Overwalle et al., 2004].
- En una más pequeña escala, una teoría de la cognición distribuida tiene aplicaciones inmediatas en negocios, gobierno y otras organizaciones. Les ayudaría a promover la innovación y evitar las desventajas de la realización colectiva de decisiones, tales como el pensamiento grupal (groupthink) Janis, 1972], que desestimulan la creatividad. Soportaría las organizaciones no sólo en la generación de nuevo conocimiento, sino en mantener, aplicar y administrar eficientemente el cono-

cimiento que ya está allí. Más fundamentalmente, nos proveería con líneas concretas para diseñar organizaciones más efectivas, donde los roles y las funciones están cláramente específicadas y donde la información es procesada en una forma coordinada, con un mínimo de perdida, distorción, malcomprensión o confusión.

- En resumen, impulsaría la inteligencia colectiva de la organización, mientras minimiza la tendencia inherente de [algunos] grupos hacia la "estupidez colectiva".

- En tecnología las aplicaciones se ven en el campo de las ontologías (Web semántica, Computación distribuida) y de la inteligencia ambiental.

La ventaja de la aproximación sugerida en el presente estudio es que usa ideas de la inteligencia artificial blanda y se basa en mediaciones y comunidades existentes, por lo cual se cree que pueden encontrarse resultados interesantes a muy bajos costos. Se cree entonces que esta investigación aporta elementos nóveles en la exploración de esta relación en la triada, comunidades, artefactos y resolución de problemas, desde la panorámica de la cognición distribuida y los sistemas multiagente.

Una pregunta no formulada en el anterior estudio, que puede explorarse desde acá estaría relacionada con el ruido que ocurre cuando las representaciones internas de los individuos se explicitan en representaciones externas, comunitariamente compartidas.

- ¿Hasta qué punto los sistemas multiagentes permiten modelar un sistema, de modo que se pueda disminuir el *ruido* entre las representaciones internas o externas y ayudar a explicitarlas o incorporarlas, respectivamente?

Sin embargo será la creación del modelo multiagente para la resolución de problemas y la exploración del mismo en una comunidad mediada por NTICs, el aspecto focal de este escrito. Se cree no obstante que los resultados tienen importante aplicaciones prácticas en el caso de ambientes de aprendizaje, comunidades virtuales y educación mediatica, que pueden ayudar a la modificación de sistemas actuales y al diseño de sistemas futuros. También se piensa que a nivel organizacional, la exploración presente puede proveer de elementos que ayuden a explicitar elementos que aumenten la eficiencia al interior de las organizaciones e instituciones, incluso si su uso de NTICs no es tan explícito o interconectado como en el caso de las comunidades virtuales.

4 Objetivos

4.1 Objetivo General

Caracterizar la resolución colectiva de problemas desde los sistemas multiagente.

4.2 Objetivos Específicos

- Diseñar y/o adaptar un modelo que permita caracterizar la interacción de sistemas multiagentes para la resolución de problemas.
- Implementar el modelo del sistema multiagente.
- Establecer un correlato entre las experiencias de aula y el modelo.

5 Marco Teórico

5.1 Preeliminares

La teoría de agentes es un desarrollo de consolidación relativamente reciente, si bien sus primeros adelantos datan de finales de los 70's y mediados de los 80's [Wooldridge, pg 304]. Minsky (1985) postuló el modelo de agentes como una herramienta teórica que permitiría modelar algunos de los problemas referidos a la comprensión de la mente. Algunos autores (Nishida, 2001) insisten en que aún no existe una definición unificada de agentes, sin embargo acá estaremos cercanos a la definición de agente Wooldridge (2001), así como su taxonomía para los entornos y su formalismo matemático para la descripción abstracta de agentes.

Definición 1. *Un agente es una entidad autónoma que se encuentra en un ambiente y actua en él con el anímo de lograr un objetivo.*

En cuanto a los entornos, estos se consideran

- Accesibles/Inaccesibles : Si para un agente toda la información del entorno está disponible, se dirá accesible, de lo contrario se considerará inaccesible.
- Determinísticos/No determinísticos : Si las mismas acciones de los agentes en el entorno producen los mismos resultados independientemente de otros factores, se llamarán determinísticos y en cualquier otro caso serán no determiníticos.
- Estáticos/Dinámicos: Si el entorno sólo cambia por la acción de un agente, se dirá estático. Si los cambios ocurren ademas por factores que se encuentran fuera de la influencia de un agente, entonces se dice que el entorno es dinámico.
- Discretos/Continuos: Un entorno discreto es aquel en el cual las acciones y percepciones sobre él son un número finito.

Existen varias taxonomías también asociadas a los agentes. Se dice que los agentes pueden ser: biológicos; robóticos (como los que se dan en la investigación y las aplicaciones industriales); Computacionales y Biológico Culturales[8]. Wooldridge señana tres características importantes en los agentes a los que vayamos a adscribir comportamiento inteligente:

- Reactividad: gracias a ella un agente responde en una manera oportuna a los cambios del entorno, a fin de lograr su objetivo.

8. En esta categoría se encuentran las criaturas de Dennet que a su vez se dividen en: Darwinianas, Skinnerianas, Popperianas y Greorianas. Estas últimas, entre las que nos encontramos, pueden usar instrumentos como la palabra.

- Proactividad: se refiere al hecho de que los agentes pueden tomar al inciativa a fin de establecer sus objetivos de diseño y mostrar un comportamiento guiado por metas.
- Habilidad Social: tiene que ver con la posibilidad de los agentes de interactuar con otros agentes.

Otros autores han discutido la movilidad y el aprendizaje como características importantes a la hora de definir agentes inteligentes, sin embargo no siempre es conveniente que los agentes aprendan nuevas comportamientos durantes su ejecución (por ejemplo en el caso de aquellos que se encargan de sistemas de misión crítica, como control de vuelo) y si bien, en general, las criaturas con movilidad son más inteligentes que aquellas no móviles (cf plantas versus animales) no es una caraterística definitoria de la ingeligencia[9].

5.2 Formalismos relacionados con agentes

Dado que un agente es una entidad situada en un entorno, empezaremos por definir el entorno formalmente para llegar así a definir un agente y avanzar posteriormente hacia los modelos multiagentes y esbozar la resolución de problemas en los mismos como un preámbulo al modelo/simulación de la segunda parte de este escrito (por desarrollar).

Miremos la primera parte de nuestra definición y su formalismo respectivo

Un agente es una entidad autónoma que se encuentra en un ambiente...

Un entorno puede estar en un conjunto discreto E de estados instantáneos que se simboliza

$$E = \{e_0, e_1, ...\}$$

obsérvese que se ha escogido un entorno como un conjunto discreto, sin embargo esto no afecta la posibilidad de que nuestros agentes operen en entornos continuos, sin pérdida de generalidad, pues supondremos que las percepciones de los agentes sobre entornos continuos, pueden ser mapeados, para efectos de su acción, en estados discretos.

En cuanto a la segunda parte tenemos

9. Para una taxonomía clara de los agentes y una revisión de las posturas al respecto se recomienda: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents* por Stan Franklin and Art Graesser en los *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996. Disponible en Internet:

`http://www.msci.memphis.edu/%7Efranklin/AgentProg.html`

Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

... y actua en él con el anímo de lograr un objetivo.

Las acciones que el agente puede hacer las definimos como el conjunto finito:

$$Ac = \{\alpha_0, \alpha_1, ...\}$$

Hemos dejado por fuera del modelo, hasta el momento, el caracter autónomo del agente y su objetivo, pero nos ocuparemos de ellos más adelante, para concentranos en el entorno y las acciones, por lo pronto. Diremos que las acciones de los agentes modifican el entorno a traves de las ejecuciones, que son secuencias intercaladas de estados y acciones, dado que cada acción de un agente sobre el estado de un entorno, produce un nuevo estado. Se tiene entonces que una ejecución r es:

$$r: e_0 \overset{\alpha_0}{\rightarrow} e_1 \overset{\alpha_1}{\rightarrow} e_2 \overset{\alpha_2}{\rightarrow} e_3 \overset{\alpha_3}{\rightarrow} ... \xrightarrow{\alpha_{u-1}} e_u$$

y, continuando con la propuesta de Wooldridge, definimos:

- $\mathcal{R}$: el conjunto de todas las posibles ejecuciones r
- $\mathcal{R}^{Ac}$: el subconjunto de $\mathcal{R}$ cuyas ejecuciones terminan en acciones.
- $\mathcal{R}^{E}$: el subconjunto de $\mathcal{R}$ cuyas ejecuciones terminan en estados.

Una función transformadora de estados τ, que representa las acciones de un agente en un entorno, puede ser vista como:

$$\tau: \mathcal{R}^{Ac} \rightarrow \mathcal{P}(E)$$

que envia un subconjunto de ejecuciones, que terminan en acciones, en un subconjunto de estados, resultado de dichas acciones sobre los estados previos.

Se define un entorno Ent como una tripleta de la forma $Ent = \langle E, e_0, \tau \rangle$, donde E es un conjunto de estados del entorno, $e_0 \in E$ es el estado inicial y τ es una función transformadora de estados.

Los agentes que habitan el entorno se definen como funciones que van de las ejecuciones que terminan en estados a las acciones. Es decir la función principal de un agente es transformar estados del entorno en acciones. Las acciones resultantes a su vez generan nuevos estados que influencian de modo posterior al agente (esta es la noción de interacción agente - entorno). Formalmente se tiene:

$$Ag: \mathcal{R}^{E} \rightarrow Ac$$

El hecho de que la función, que define al agente, tenga en cuenta las ejecuciones que terminan en estados y no sólo los estados, quiere decir que los agentes tienen algún sentido de la historia (salvo en el caso de los agentes púramente reactivos, que mencionaremos brévemente más adelante).

Ahora bien, los estados del entorno no son equivalentes a lo que el agente «sabe» de éste, pues en general los agentes habitan entornos inaccesibles en toda su información. Esto introduce la noción de percepción. Una percepción es aquello que el agente capta de los estados del entorno (que no forzósamente tiene que ver con los estados reales) a través de una función sensora

$$\text{sensor}: E \rightarrow \text{Per}$$

y las acción es un mapeo que van desde las percepciones a las conjunto de acciones:

$$\text{accion}: \text{Per} \rightarrow Ac$$

un agente entonces es un par sensor-acción, $Ag = \langle \text{sensor}, \text{accion} \rangle$.

La literatura de los agentes los divide en varias clases, de acuerdo al enfoque de modelaje que se haya asumido y las propiedades pertinentes para cada uno. Wooldridge distingue:

- **Agentes de razonamiento deductivo**: Aquellos que se basan en la lógica deductiva (casi siempre mediante predicados en lógica de primer orden) para decidir y modelar sus interacciones con el entorno. La crítica a estos se basa en que no siempre se usa sólo la lógica para determinar el curso de una determinada acción en un entorno, por tanto el modelo puede no ser tan fiel a la realidad.
- **Agentes de razonamiento práctico**: El razonamiento práctico busca balancear posturas encontradas y a favor de determinadas acciones de los agentes en el entorno. Wooldridge (2001) señala que el razonamiento práctico es la deliberación más el razonamiento orientado por fines y medios. La deliberación se ocupa de preguntarse *qué* es lo que se quiere hacer, mientras que los fines y medios se ocupan de *cómo* hacerlo. Los agentes de razonamiento práctico se basan en las ideas filosóficas de Bratman (1990, citado por Woldridge) y fueron desarrollados por [?]:
- **Agentes reactivos e híbridos**: En este enfoque se asume que la inteligencia es un comportamiento emergente de sistemas complejos que no tienen porque tener un modelo explícito de razonamiento. Su más vocal exponente es Roodney Brooks con su idea de la *arquitectura de subsumción*, en la que diferentes comportamientos, en capas inferiores subsumen comportamientos en capas superiores. A pesar de su elegancia, simpleza y tolerancia a fallas, ha encontrado dificultades particularmente referidas al diseño empleando esta arquitectura por la difícil previsión de muchos factores, por ejemplo la relación de las acciones locales y los comportamientos globales.

5.3 Interacciones multiagente

Nos interesan, sobre todo, las interacciones que puedan ser modeladas desde representaciones reactivas en los agentes. Por esto nos apartaremos de los enfoques "clásicos" de las interacciones multiagente desde la teoría de juegos, si bien algunos de sus constructos teóricos como equilibrio de Nash, utilidad, etc., pueden sernos de ayuda a la hora de enfrentar nuestro tema.

Un ejemplo clásico de las interacciones multiagente es el del dilema del prisionero[10], citado ampliamente y sobre el cual no haremos mayor incapie, salvo para destacar el hecho de que en una circunstancia como la que este dilema plantea, la traición parece ser la mejor opción. Sin embargo en el dilema del prisionero iterado, donde los encuentros en circunstancias similares se dan una y otra vez, la cooperación surge "naturalmente", si se tienen en cuenta las interacciones a largo plazo. El modelo del prisionero iterado ha sido extendido a campos particulares dónde además se consideran las opciones de cada un o de los jugadores de acuerdo a ecuaciones que describen las variables del entorno en el cual se encuetran situados, por ejemplo para considerar la modularidad del código que se produce por un conjunto de programadores y s i se debe revelar o no el código propio (Baldwin, 2003). Existen otro tipo de interacciones simétricas de dos agentes, cuyas variaciones se dan de acuerdo a la percepción de la función de utilidad de cada una de las acciones de los involucrados (Wooldridge, 2002 pág 123), sin embargo nos preocuparemos de las interacciones de más de dos agentes.

Las interacciones multiagente fueron una de las influencias del Modelamiento de Sistemas Basado en Agentes junto con otras fuentes teóricas (Macal, 2005) como: teoría de juegos, Inteligencia Artificial, Sistemas Complejos Adaptativos, Autómatas Celulares, Teoría de la Gestión, entre otras. Sus orígenes se deben al trabajo del físico Stanislaw Ulam con autómatas celulares, como una forma de dar espuesta a uno de los problemas propuestos en el siglo en el siglo XX por el matemático Jon von Newman, acerca de si podrían fabricarse máquinas capaces de auto reproducirse (y la respuesta fue de hecho sí).

10. Brevemente el dilema del prisionero considera dos criminales que han sido capturados pero aún no han determinado su culpabilidad. A cada uno de ellos se le interroga por separado y se le pide que considere las siguientes opciones: que los dos se queden callados, caso en el cual a cada uno se le da dos años de cárcel, que uno delate al otro mientras el otro está callado, caso en el cual el soplón sale libre y al delatado le dan 5 años de carcel y que los dos confiesen, caso en el cual cada uno tendrá tres años de prisión. Este es un ejemplo clásico que aparece en la gran mayoría de libros de teoría de juegos. Al lector interesado se le recomienda en particular a Axelrod y su libro La complejidad de la cooperación (fondo de cultura económica, 2004), para una presentación básica del modelo y una extensión del mismo en los casos iterados.

John Holland identificó las propiedades básicas de los sitemas adaptativos complejos y sus mecanismos[11] y su trabajo lo condujo al desarrollo de los algoritmos genéticos. Posteriormente el matemático John Conway desarrollo un algoritmo llamado el juego de la vida en el cual un conjunto de reglas simples, en la que se usaba información disponible localmente a los autómatas en una grilla, generaba comportamientos globales que se mantenían y eran altamente sensibles a las condiciones iniciales en las cuales estaban los atuómatas.

Stephen Wolfram demostró que se podrían establecer correspondencias entre un amplio rango de patrones emergentes en los autómatas celulares (obtenidos al cambiar sus reglas) y diferentes sistemas lógicos y un gran número de algortimos. Pero el modelamiento basado en agentes no solo se usó para entender sistemas lógicos, formales o sistemas artificiales, sino que un importante campo de aplicación ha sido el modelamiento de comprotamientos biológicos, sociales y cognitivos, entre otros. El primer modelo social se aacredita a Thomas Schelling quien aplicó las nociones de autómatas celulares para modelar la segregación racial y la conformación de *gethos* como propiedades emergentes que no estaban implicadas ni eran consistentes con la programación individual de los agentes. Los sistemas multiagente también han sido utilizados por los antropólogos para modelar los factores que causaron la desaparición del los Anazasi es Norte América (Koehler et al. 2005, citado por Macal 2005) y la caida de los pueblos de la civilización Mesopotámicos (Christiansen y Altaweel 2004, citado por Macal 2005). También se ahn usado en sociología para modelar las relaciones entre individuos y en economía para la exploración de agentes no tan ideales como los de los modelos tradicionales, donde, por ejemplo, el equilibrio a largo plazo no es tan importante como la exploración de estados intermedios, o donde se supone que la optimización no es un objetivo de todos los agentes, sino que basta con la "satisfacción" (concepto introducido por Herbert Simon) o que los agentes no son homogéneos.

Especial consideración merecen los abordajes multiagentes hechos por la ciencias cognitivas, dado que este enfoque es el que particularmente nos atañe en este trabajo. Se están

11. Las propiedades son: 1) La agregación permite la formación de grupos, 2) La no linearidad invalida la extrapolación simple, 3) los flujos permiten la transferencia y la transformación de recursos e información y 4) la diversidad permite a los agentes comportarse diferentemente entre sí y a menoduo permite la propiedad del sistema de robustez.

En cuanto a los mecanismos estos son: 1) etiquetamiento: que permite a los agentes ser nombrados y reconocidos, 2) los modelos internos le permiten a los agentes razonar sobre sus mundos y 3) los bloques de construcción les permiten a los sistemas y a los componentes se compuestos de componentes más sencillos. Este último mecanismo inspiró mucho del diseño orientado a objetos.

creando modelos basados en agentes para la emoción, la cognición y el comportamiento social. En palabras de Macal (2005) «el objetivo es crear agentes sintéticos que encarnen el matizado interjuego entre emoción, cognición y comportamiento social».

Uno de los trabajo seminales en el campo de la cognición y los sistemas multiagentes fue el de Marvin Minsky con su libro Society of Mind (Minsky, 1986) en el cual él plantea que la comprensión de la mente no se puede hacer desde principios simples universales aplicables a toda ella, sino de la diversidad de los múltiples agentes que se unen en agencias complejas[12]. En este escrito Minsky ofrece un modelo que permite entender diversos campos de la actividad mental como la visión, la formación de conceptos, el lenguaje, el humor, las emociones entre otros (el tema de la emoción es considerado a fondo en su texto próximo a salir *The Emotional Machine*) y enmarca un derrotero investigativo en el cual aún hay muchas cosas por aprender y explotar, incluyendo las aplicaciones prácticas y la construcción de modelos computacionales que empleen los constructos presentados. Un análisis detallado de la teoría presentada en dicho libro está por fuera de los alcances de este texto (para una revisión panorámica se recomienda Sing 2003), sin embargo, es de anotar que en los planteamientos hechos pretenden explicar la mente, pero no tienen en cuenta al observador en las interacciónes que éste se plantea con artefactos y mediacones culturales: libros, computadores, otros seres humanos o el modelo mismo de *Society of Mind*. En este sentido el correlato educativo del trabajo acá presentado pretende ser un aporte, puesto que no es necesario tener una implementación computacional completa del modelo congitivo en un sistema multiagente, o incluso de partes del modelo (como la resolución de problemas), para que artefactos y creadores se empoderen mutuamente en contextos culturales. Si bien el modelo de Minsky no hace aluciones a la mente colectiva y las experiencias son referidas a individuos, creemos que lo educativo puede ser el puente que vincule a las mentes individuales y colectivas con los artefactos culturales (incluidos los modelos formales y computacionales que crean dichas mentes para entenderse y modificarse a sí mismas).

En todos los anteriores modelos se parte de la premisa de que el fenómeno que se desea modelar puede ser creíblemente modelado a un nivel razonable de abstracción para un propósito particular. Dado que la unidad de análisis es el colectivo de los agentes, los fenómenos emergentes en el colectivo son los que interesan y se pueden hacer varias sim-

12. El término agente lo uso para describir cualquier componente de un proceso cognitivo lo suficientemente simple como para ser entendido y el término agencia para designar sociedades de tales agentes que juntos realizan funciones más complejas de lo que cualquier agente individual podría hacer.

plificaciones respecto a los agentes individuales o entrar en gamas que los hagan más cercanos a los agentes reales, dependiendo de los recursos disponibles para el modelo (tiempo, capacidad de computo, complejidad de programación) y el objetivo del mismo. La pregunta entonces por la credibilidad lleva a cuestionamientos sobre la validez de los modelos multiagente y se han propuesto, esencialmente, dos formas: la validación comportamental y la validación funcional. En la primera se recolectan los datos de la realidad que pueden ser introducidos en el modelo y luego de que la realidad y el modelo avanzan por su cuenta, se comparan los datos nuevamente, si la simulación afectó los datos de entrada, de modo que corresponden a los datos reales cuando el tiempo ha avanzado, el modelo se considera válido. Esta forma de validación suele ser la más costosa, en términos metodológicos y computacionales, cuando la escala y la complejidad del fenómeno, de los cuales quiere dar cuenta el modelo, crecen. En la segunda, la validación funcional, se busca apreciar si el funcionamiento del modelo se asemeja o da cuenta del funcionamiento de la realidad, para lo cual se miran los fenómenos emergentes en el uno y la otra. La ventaja de la validación funcional es que los cuestionamientos de escala no son tan importantes, ya que los fenómenos emergentes pueden ser vistos a diferentes escalas, y, por tanto, requiere menores recursos de computo. Esta será el enfoque con el que nos aproximaremos a la validación del modelo presentado en este escrito, así que consideraremos con mayor detenimiento la propiedad de emergencia en los sitemas multiagentes.

Aún no existe una definición uniformemente aceptada de emergencia en los sistemas complejos. Se la considera como (Pfeifer y Scheier, 1999): una propiedad de un fenómeno o del sistema complejo que no es completamente entendida; como una característica que se encuentra en el sistema como un todo pero no en sus partes individuales y como el resultado de la interacción entre el agente y el entorno que no fue programada en el modelo. Estas definiciones involucran al observador, pues es él quien distingue entre las partes y el todo y las características de cada uno, y también le atañe al observador la apreciación de las interacciones entre el entorno y los agentes que lo habitan, así como de la estructura algorítmica que da cuenta de dichas interacciones, para determinar si el comportamiento estaba programado o no. Müller (2002, referenciado por Phan 2004) pone de manifiesto el papel del observador en la definición de emergencia puesto que «subraya la necesidad de acoplar el proceso con el nivel de observación del proceso» y distingue dos tipos de emergencia: la débil, que es detectada por observadores/sensores externos al sistema y la fuerte, que es detectada por observadores/sensores internos al sistema[13]. Lo interesante de

la definición de Müller, en términos de sensores, es que no se prejuzga la naturaleza del observador y este puede ser natural o artificial (sólo basta con que se disponga de sensores para percibir el fenómeno) y además es coherente con la definición desde los sistemas complejos en términos de una propiedad que está en el sistema pero no en sus partes individuales. Si los elementos que perciben la emergencia están dentro del sistema (emergencia fuerte) esto implica la posibilidad de reflexionar o volver sobre el sistema (similar a los cerebros-B de la teoría de Minksy, que pueden percibir los estados de los cerebros-A): Esto tiene un interesante correlato metacognitivo, en palabras de Phan:

«El hecho de que un observador artificial pueda detectar un fenómeno emergente implica que (1) el sistema en sí mismo o (2) los elementos que lo constituyen sean capaces de retroactuar sobre el proceso. El primer caso emerge cada vez que detectamos nuestros estados mentales claramente. Como una comunidad neuronal, nuestro sistema nervioso es capaz de detectar la entrada en actividad de carias clases de poblaciones particualres de sus propias neuronas. El segundo caso surge cuando el sistema consiste de agentes cognitivos. Aquellos son entonces capaces de localizar ciertos requerimientos del sistema en el cual ellos están inmersos (emergencia fuerte en el significado de Müller)».

Es decir, la emergencia asociada al observador implica una forma conciencia del sistema multiagente sobre sí mismo.

5.4 Resolución de Problemas en sistemas multiagente

La resolución colectiva de problemas se ha abordado en la Teoría Computacional de la Organización y en la resolución de problemas distribuída. Desde el primer abordaje, se puede decir que, al igual que las organizaciones[14], la resolución colectiva de problemas surge como una forma de sobrepasar las limitaciones cognitivas, físicas, temporales o institucionales de la agencia individual. Desde el segundo, existen varias motivaciones para la resolución de problemas distribuída (Durfee Weiss 1999): (1) que gracias al paralelismo de

13. Müller habla formalmente de un lenguaje D, en el cual se describen las dinámicas del sistema agente-entorno y de que se requiere un lenguaje diferente a D para describir la emergencia (ya sea fuerte o débil). Se trata entonces de un *metafenómeno*, puesto que los agentes que participan de un fenómeno se "dan cuenta" que está ocurriendo otro, en un divel distingo, pero el lenguaje que describe al perimero no basta para hablar del segundo. El sistema requiere de capacidades de introspección y reflexión.

14. Tampoco existe acá una definición formal de organización ampliamente compartida, pero sí una caracterización como (1) tecnologías resolutoras de problemas a gran escala, (2) compuestas de múltiples asgentesarticifiales, naturales o ambos (3) sistemas de actividad involucrados en una o más tareas, (4) poseedoras de historia, cultura, memorias y capacidades distintas a las de cualquier agente, (5) capaces de afectar y ser afectadas por el entorno. En este sentido el aula es una organización pequeña.

recursos se puede agilizar la resolución del problema, (2) que las capacidades para la resolución estén inherentemente distribuídas (3) que la planeación o los resultados de la resolución del problema puedan ser distribuídos para ser ejecutados por diferentes agentes.

Sean cuales sean las motivaciones que dan origen la resolución colectiva de problemas, este fenómeno esta permanentemente involucrado a nuestro mundo y hace parte esencial del entramado cognitivo y social y, si se quiere, estructura, de los fenómenos complejos, por lo cual se han concebido varios programas de software y diseños formales (Plural Soar, Sugarscape, HITOP-A, Swarm, Jade, Jadex, AgentSpeak-L), que colocan varios atributos indivuales en los agentes, a fin de hacerlos más adecuados dominios del problema específico que se desea modelar, haciéndolos más completos y complejos. Sin embargo, en la medida en que el modelo es más general y abstracto, como en el caso de esta tesis, los agentes se hacen más genéricos, sencillos y reactivos y los modelos multiagentes generales de resolución de problemas, se asimilan mucho a modelos de búsqueda distribuída, pues, además de que los modelos computacionales de búsqueda son bien entendidos, se supone que la resolución de problemas es una búsqueda en un espacio de soluciones, realizada por un sistema multiagente, pues, en caso de que la solución se conocierá, no habría problema a resolver (Carley y Gasser, compilados por Weiss 1999).

Acá asumiremos en enfoque de los agentes reactivos como forma de modelar la resolución de problemas colectiva, en particularmente por la simplicidad algorítmica de implementación y por el hecho de que nuestra unidad de análisis no es el agente individual, sino el colectivo de agentes y usaremos la hipótesis de Brooks respecto a la inteligencia, y *eventualmente la resolución de problemas*, como una propiedad emergente del sistema multiagente. Es decir, agentes reactivos pueden dar cuenta de propiedades en un colectivo que se considera inteligente, sin entrar a considerar qué tan inteligentes son las motivaciones detrás de las reacciones individuales de los agentes. En ese sentido se ha elegido a agentes reactivos sobre agentes a los que se pueda asociar estados mentales (creencias, deseos e intensiones), que expliquen sus interacciones entre sí y en el entorno. El modelo de creencias, deseos e intensiones (BDI, por sus siglas en inglés), es uno de los más maduros y durables en la literatura generalizada de agentes y que cuenta con bastantes implementaciones de software y desarrollos teóricos que lo soportan. Es de anotar, sin embargo, que este abordaje elegido, desde agentes reactivos, no implica que los entornos en los que la resolución colectiva de problemas se modela, no puedan estar habitados por otro tipo de agentes, por ejemplo por agentes razonamiento. En los anexos se ofrecen algunos elementos de resolución de problemas desde el modelo Creencias, Deseos e Intensiones.

6 El modelo

6.1 Problema

Un problema se definirá en el modelo como una diferencia entre el estado objetivo de un entorno multiagentes y el estado actual del mismo y la resolución del problema será la ejecución de unas acciones cuya finalidad sea hacer coincidir el estado futuro del entorno en el objetivo y el estado percibido como actual[15].

Obsérvese que, por lo pronto, no estamos asumiento el interrogante por la creación de planes o la planeación, a pesar de que la resolución del problema es, en nuestra aproximación, equivalente a la ejecución de un plan. Si bien la literatura sobre agentes considera el asunto de la planeación (por ejemplo HOMER, citado por Wooldridge, pág 80, o los lenguajes de planeación de agentes como 3APL , referenciados porDastani, sin año), en este escrito podremos asumir que los planes pueden ser preprogramados por un agente humano en un agente específico y que este puede seleccionar un plan prediseñado de una pila de planes a elegir[16]. Nos interesa más cómo las acciones individuales en dichos planes se ponen en ejecución y se comunican, en ambientes multiagente.

En términos de las notaciones antes establecidas diremos, entonces, que un problema P es la diferencia entre un estado e_0 y un estado e_f del entorno E. A su vez el estado e_0 está caracterizado por un conjunto ordenado de valores $e_0 = \{v_1, v_2, ..., v_n\}$ y el estado e_f esta caracterizado por un conjunto ordenado de valores $e_f = \{v'_1, v'_2, ..., v'_n\}$. Se dice que el problema persiste si existe al menos un $v_i \neq v'_i$ para i dentro de los índices de los estados e_0 y e_f y se dice que el problema fue solucionado si $v_i = v'_i$ para todo i en los índices de los estados e_0 y e_f. El tamaño del problema es la cantidad de valores para los cuales $v_i \neq v'_i$. Es decir, un problema solucionado es aquel que tiene tamaño 0.

6.2 Características para la resolución y algoritmo para los agentes

El sistema multiagente se caracterízará por tres elementos para la resolución:

- Coherencia (*querer resolverlo*): «La propiedad o estado de actuar como una unidad. La coherencia tiene que ver con qué tan bien se comporta el sistema como una unidad. Los criterios de evaluación para la coherencia son, por ejemplo, eficiencia,

15. En este sentido se puede decir que nuestro sistema multiagente resolutor de problemas es un motor-diferencial bajo el enfoque de Sociedad de la Mente de Minksy.

16. Es decir, dentro de nuestros sistemas de razonamiento práctico nos concentraremos en lo que llaman sistemas de planeación reactiva, donde los planes vienen prediseñados dentro del agente, si bien no descartamos la posibilidad de usar sistemas de planeación deliberativa como 3APL, eventualmente.

calidad de la solución y degradamiento elengante en presencia de fallas» (Huhns, Michael N. y Stephens, Larry M., 1999). Obsérvese que si bien la coherencia es una propiedad emergente del sistema, se requiere que cada agente «quiera» actuar como unidad, de acuerdo a la labor que le fue asignada. Es decir la coherencia grupal surge del aporte individual de los agentes.

- Competencia (*saber resolverlo*): «Es la habilidad de hacer una tarea bien» (Durfee, Edmund, H., 1999). La coherencia sólamente no basta para resolver un problema, sino que se requiere que cada uno de los agentes individuales esté en condiciones de resolver pla parte del problema que le fue encomendada. De nuevo, se trata de una propiedad global que surge de los aportes individuales de los agentes.
- Modularidad (*descomponible en partes acopladas*): «Es referida como una propiedad general de los sistemas complejos, concerniente al grado de *descomponibilidad* del sistema en subpartes vagamente acopladas hechas componentes fuertemente acoplados. [...] Un sistema modular, entonces, es representado como un complejo de componentes o sub-sistemas donde los diseñadores tratan de minimizar y estandarizar las interdepenencias entre los módulos» (Schilling, 2000; citado por Narduzzo y Rossi).

Obsérvese que, mientras las dos primeras características están referidas a los agentes individuales, la última se refiere al problema, que en este caso es modelado por el entorno de los agentes, y por tanto también es una característica referida a la relación e interacción entre los agentes y el problema.

Es de anotar también que los agentes no habitan un espacio físico, sino un espacio formal, donde se modela el problema. El entorno y los agentes admiten representaciones gráficas bidimensionales a fin de visualizar el modelo, sin embargo se intenta capturar las características genéricas de un problema en un espacio formal y de los agentes que lo han de resolver.

Las características de competencia y coherencia, si bien son poseídas individualmente por los agentes, pueden ser leídas desde premisas asociadas a ellos como pertenecientes a un colectivo. Es decir, determinados niveles de coherencia y compentencia individuales, pueden dar cuenta de un conjunto de valores o *ethos* y saberes comunes, que a su vez pueden dar cuenta de fenómenos complejos, como el tamaño de la cultura compartida[17].

17. Suponemos acá un acercamiento similar al que hace Axelrod (2004) para la cultura, es decir, la idea de que para efectos de un sistema multiagente, las cultura puede ser modelada como un conjunto discreto de variables que a su vez toma un conjunto discreto de valores. Dos agentes poseen la misma cultura si la misma variable tiene

Los agentes, entonces, tendrán indices de coherencia y competencia que podrían dar cuenta de que todos tienen intensiones o saberes similares, o bien, que a pesar de ser saberes distintos, son complementarios en la resolución del problema. El hecho de que los agentes se comuniquen a través de las percepciones y modificaciones del entorno también da cuenta de fenómenos presentes en el sistema, más que en los agentes individuales, es decir asociadas a la interconexión e interdependencia, que a su vez podrían ser asociadas a propiedades no explícitas en el modelo, como las culturales.

Conservaremos acá la definición de agente como una función transformadora, presentada anteriormente, pero la afinaremos para dar cuenta de las características de coherencia, competencia y modularidad. Decíamos que un agente era una función

$$Ag: \mathcal{R}^E \rightarrow Ac$$

de $\mathcal{R}^E$, el subconjunto de las ejecuciones $\mathcal{R}$, cuyas ejecuciones terminan en estados, en el conjunto de acciones Ac. Esta definición presentaba el dominio y el rango de la función, pero no la definía de modo explícito (puesto que era una definición general). Para efectos del modelo, se procederá a definirla acá de forma específica a partir de cómo el agente «tomará una decisión» sobre si cambiar o no el entorno a partir de sus niveles de coherencia y competencia.

Este es el algoritmo del agente para cada ciclo de ejecución:

- Tamaño del problema > 0 ?
    - Si:
        - Nivel de coherencia está por encima de un mínimo prefijado?
            - Si:
                - Es competente para aportar a la solución de ese módulo del problema?
                    - Si:
                        - Aporta a la solución cambiando el valor del estado del entorno en donde se encuentra ubicado.

el mismo valor. Es de anotar, sin embargo, que el modelo presentado en este trabajo no coloca a la cultura en una estructura de datos de los agentes y/o del entorno de modo explícito, sino que la modela implícitamente a patir de los niveles de coherencia y competencia de los agentes, así como de modularidad del problema.

- Disminuye el tamaño del problema en una unidad.

– No:

- Se desplaza a otro «lugar» del estado del entorno.

• No:

◦ Se desplaza a otro «lugar» del estado del entorno.

◦ No

– Se detiene

6.3 Implementación

Para la implementación del modelo se escogió Kedama (Yoshiki, sin fecha), un sistema de simulación multiagente, implementado en Squeak, que se ha empleado para realizar modelos con agentes reactivos, ya sean estos biológicos, como en las colonias de hormigas (Yoshiki, sin fecha), o sociales, como la segregación de Schelling (Guillaume, 2006), entre otros. Las razones de su escogencia no sólo están asociadas a su adecuación para la simulación que se pretende, sino, además, al hecho de brindar un sistema de programación visual vía mosaicos (*eToys*), de poseer un lenguaje subyacente orientado a objetos de sintaxis simple y elegante y de ser Squeak el sistema con el cual se estableció el correlato y la experiencia de aula y educativa del modelo, cuya implementación se muestra en esta sección. En el anexo 4 se presenta una introducción a la arquitectura de Kedama, en este punto sólo se presentará la parte inicial, que permite entender la forma en que se implementó el modelo en dicho sistema, para una referencia más detallada se recomienda leer el anexo y la documentación de la Simulación de Schelling[18].

Kedama está hecho de varias partes. En la parte inferior está el mundo Kedama en sí mismo, el cual aparece como una gran caja negra. Encima de estos está una grilla invisible de cuadrados. Cada cuadrado es llamado un patch (parche), y la grilla entera es controlada por variables parche que puedes programar. Los parches sólo son encontrados en

18. Puede encontrarse un texto introductorio a dicha documentación del modelo de segregación de Schelling en español en: `http://www.el-directorio.org/Kedama/SegregacionSchelling`

Kedama y pueden ser pensados como de un papel de gráficos con 100 x 100 celdas (10.000 en total). A cada celda puede ser asignado un valor numérico. Moviéndose a través de esos parches en el mundo Kedama están los rebaños de tortugas, que son grupos de tortugas que actuan en la misma forma. Estas tortugas, moviendose a lo largo del mundo Kedama, pueden también leer el valor de cada parche y responder a él. Finalmente, hay tortugas individuales, o partículas, que comparten en mismo nombre que su rebaño. Por ejemplo, si el primer rebaño que se crea es `tortuga1`, así que cada tortuga en este rebaño seguirá las instrucciones a la `tortuga1`. A las tortugas se les da las instrucciones como rebaño, no como tortugas individuales.

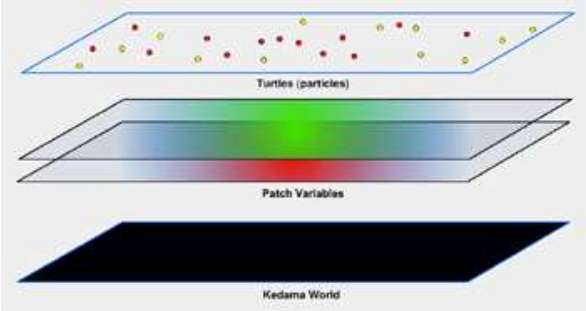

Figura 1. Arquitectura de capas del mundo Kedama

Se colocaron dos mundos Kedama, llamados `Kedama` y `Kedama1` que representan el estado final y el estado original en el que se encuentra el entorno que habitan los agente. Como se dijo en el modelo formal, la diferencia en estos estados es lo que constituye un problema. A su vez cada problema puede ser dividido en subproblemas, que en el mundo Kedama son representado por parches (*patchs*). Para efectos de esta implementación se colocó un problema que constaba a su vez de tres subproblemas o módulos. Se eligió una métrica sencilla para la modularidad diciendo que la modularidad es igual al número de módulos.

Se definieron tres variables en el mundo que representa el estado objetivo: `problemaTotal`, `problemaActual` y `solucionActual`, el primero mide el tamaño del problema, es decir la cantidad de valores del estado objetivo que se desean que sean diferentes a las del estado actual, el segundo el tamaño del problema cuando se está planteando, y el tercero mide el tamaño

Se programaron dos tipos de agentes (tortugas, en la terminología de Kedama), unos que recorrían el mundo planteando diferentes tipos de problemas y otros que los solucionaban. El algoritmo método principal del agente que plantea el problema se muestra a continuación:

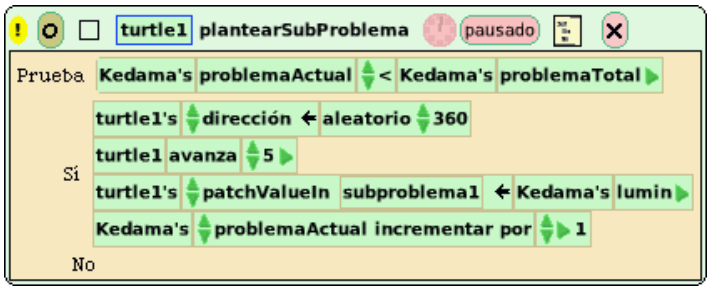

Figura 2. Método de los agentes que plantean el problema

Su funcionamiento es como sigue: El primer tipo agente compara si el problema que el tiene que ayudar a plantear ya es del tamaño del problema deseado, en caso de que no lo sea (es decir que `problemaActual` sea menor que `problemaTotal`) elije un lugar aleatorio dónde ubicarse y escribe en el *patch* `subproblema1`, que representa un módulo del problema que está ayudando a plantear, y cambia el valor del *patch*, incrementando luego el `problemaActual` en una unidad. De modo análogo ocurre con los otros dos tipos agentes que plantean los otros subproblemas. Nótese que aquí hemos hablado de tipos de agentes y no de agentes específicos, dado que podemos colocar a varios agentes de un sólo tipo a que ayuden a plantear un subproblema, caso en el cual lo valores diferentes entre el estado inicial del entorno y el estado objetivo estarán más exparcidos por todo el entorno.

La resolución del problema es la que implementa en Kedama el algoritmo de la sección anterior y el núcleo de esta tesis. Se han colocado dos tipos de agentes para solucionar los tres subproblemas, unos, los verdes, que sólo puede solucionar subproblemas del tipo 3 (o planteados en espacio del *patch* 3) y otros, los rojos que puede solucionar problemas del tipo 1 o 2 (planteados en los espacios de los *patch* 1 y 2). En la terminología del modelo los agentes rojos son más competentes que los agentes verdes. Siguiendo la hipótesis de Narduzzo y Rossi (2003) en cuanto a que problemas más modulares afectan positivamente el deseo de ser solucionados por los agentes en un colectivo (pues se pueden encargar de secciones más pequeñas), se ha puesto un factor de aleatoriedad, que modela la coherencia y se relaciona con la modularidad de este modo: si el número aleatorio que modela la coherencia es menor que la modularidad entonces el agente intentará verificar su competencia (si puede aplicar su saber a la solución) y en caso de que sea competente y el espacio solución aún esté sin solucionar, aportará a la solución cambiando el estado en el *patch* que modela la subsolución e indicando esto mediante el incremento de la variable `solucionActual` en una unidad. El algoritmo, implementado en Kedama, es como sigue:

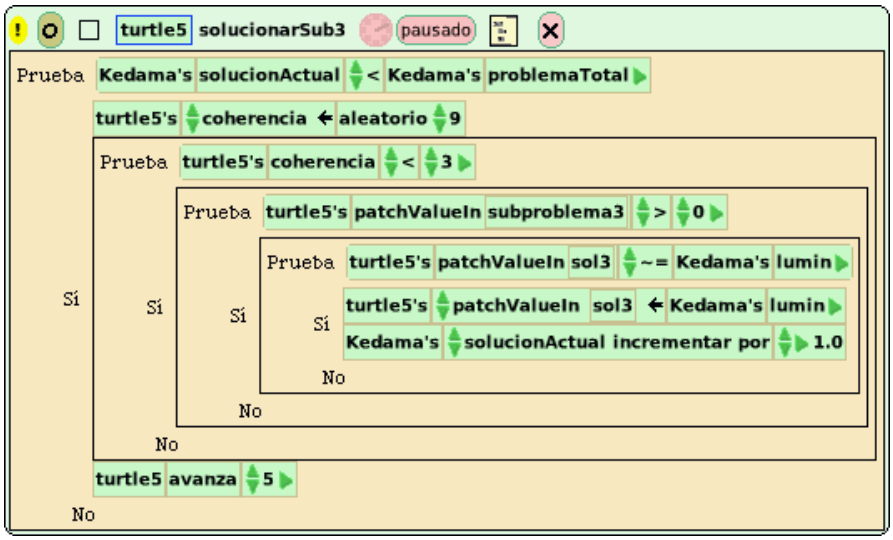

Figura 3. Algoritmo de solución de problemas del modelo presentado en esta tesis implementado en Kedama. Se deben usar pruebas anidadas para simular el operardor "*y*" lógico, pues el sistema de programación gráfico *eToys* de la versión actual de Kedama no lo soporta en la programación gráfica que se emplea.

El algoritmo de los agentes rojos es similar en esencia, con la diferencia de que, al ser más competentes, deben hacer más verificaciones anidadas, asociadas a dónde pueden aplicar su saber (si en el subproblema uno y/ó en el subproblema 2). La implementación en Kedama se muestra a continuación:

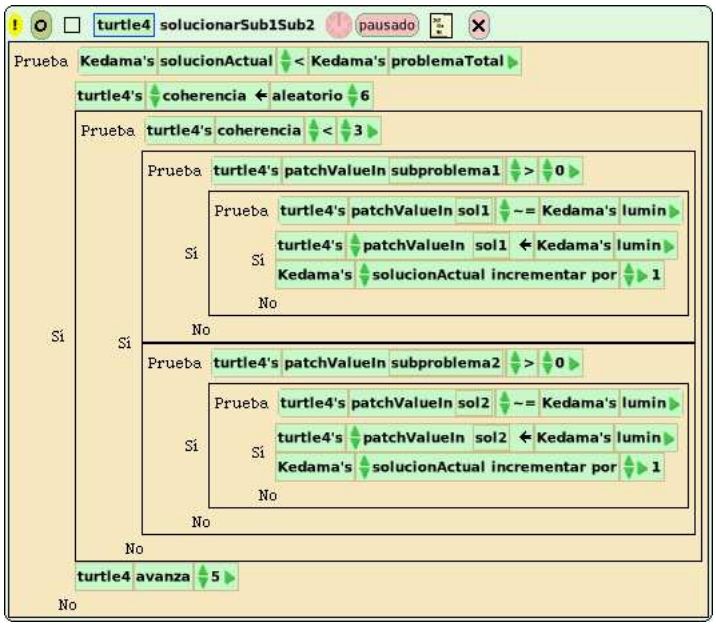

Figura 4. Implementación del algoritmo del modelo, para agentes más competentes

Finalmente se implementó en en entorno Squeak una interface con un conjunto de controles que permitieran correr varias simulaciones cómodamente, especificando el tamaño del parámetro `problemaTotal`, y controlar los momentos en los cuales se invocaban los métodos de planteamiento de problemas y ejecución de soluciones por parte de las familias de agentes. Unas captura de pantalla del entorno se muestran a continuación:

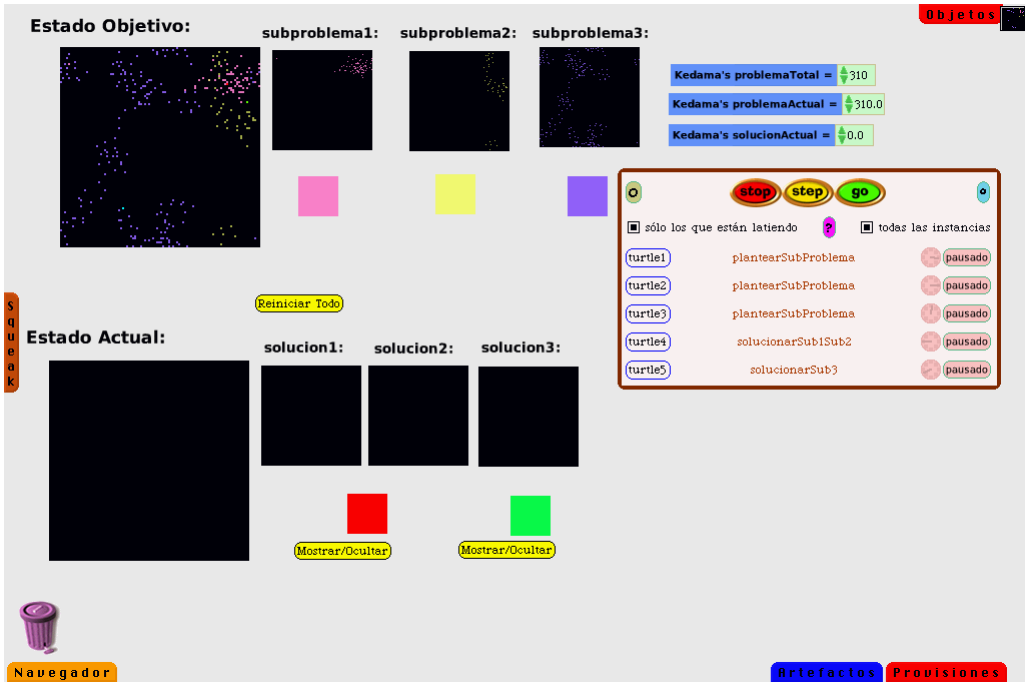

Figura 5. El entorno de ejecución de las simulaciones del modelo. Se puede ver el estado objetivo del entorno, constituido a su vez por tres subproblemas o módulos y, abajo de cada uno, ejemplares de las familias de agentes que ayudan a plantearlos. A la derecha están los valores de las variables más importantes del entorno y una utilidas que permite invocar la ejecución individual de los métodos para plantear subproblemas o sus soluciones. En la parte inferior se puede ver el estado actual del entorno, antes de invocar la solución del problema, junto con las subsoluciones y los ejemplares que las resuelven. Para una imagen de mayor resolución, ver los anexos.

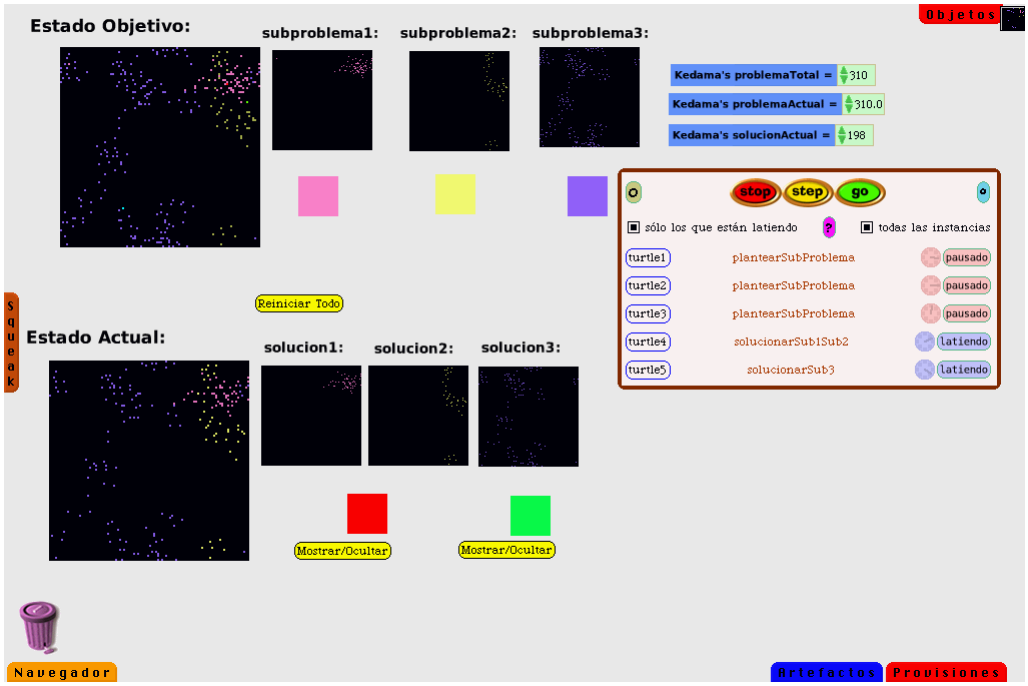

Figura 6. Simulación ejecutando los algoritmos de resolución del modelo, lo que se aprecia porque los relojitos que controlan a las familias de agentes rojo y verde tienen la leyenda "latiendo" en la utilidad de control de guiones a la derecha y están de color azul. Obsérvese cómo las soluciones 1 a 3 de la parte inferior empiezan a parecerse a los subproblemas del estado objetivo en la parte superior. Pra una imagen de mayor resolución ver los anexos.

7 Correlato educativo del modelo

El correlato educativo del modelo y el modelo mismo se desarollaron en paralelo durante un tiempo de tres semestres (dos del 2006 y el primero del 2007). La intensión durante este tiempo fue poner a charlar tanto a la experiencia de aula como a la teoría de sistemas multiagentes para que se inspiraran mutuamente. Esta sección pretende dar cuenta de los momentos más relevantes de dicho diálogo, identificando los hitos más importantes durante la construcción del modelo.

Primer momento: No hay coherencia, competencia, ni modularidad

Durante el primer semestre de la experiencia, el modelo no era claro, así que se pretendió observar el surgimiento del problema colectivo a partir la configuración de una mediación digital que sirviera como un espacio común para el nacimiento de dichos problemas. Para esto se configuró y puso a disposición de los estudiantes y profesores del curso de Introducción a la Informática, un sistema colaborativo y con memoria de creación de documentos hipertextuales, llamado Eduwiki[19]. Se les solicitaba a los estudiantes que escogieran un problema de su interés para la realización durante el curso, que fueran publicando una bitácora de avance sobre ese proyecto y que leyeran y aportaran a las bitácoras de sus compañeros y se esperaba que, en la medida en que problemas distintos involucraran conceptos algorítmicos iguales (ciclos, condicionales, etc.) empezaran a surgir problemas colectivos sobre tales conceptos, cuya raíz eran los problemar particulares y diferentes de cada estudiante.

Este esquema no tuvo éxito. Los estudiantes no se leían entre sí ni aportan consistentemente a las páginas de sus compañeros, por lo cual el problema colectivo no surgía. A esto se aunaba el hecho de que, al ser introducción a la informática una materia electiva y de matrícula abierta, se contaba con estudiantes de distintas carreras (ingeniería de sistemas, nutrición, biología, informática matemática) y distintos semestres, con preocupaciones diferentes que se reflejaban en la elección de sus problemas específicos. A pesar de que existían sugerencias sobre los lenguajes de programación y entornos a utilizar, con una

19. Existen muchas variantes de Wikis, en su concepción original los wikis son sólo sistemas de creación rápida de hipertexto, pero el caracter colectivo y con historia es lo que los ha hecho tan pertinentes en los escenarios de gestión de conocimiento, documentación empresarial y ha generado el éxito del proyecto Wikipedia, por ejemplo. La dirección del wiki educativo es `www.eduwiki.info`

predilección por los de sintaxis sencilla (Python, Smalltalk/Squeak), el hecho de que existiera tantas variantes temáticas y de implementación aunado a una práctica en la cual no se leía a los compañeros hizo extremadamente difícil el surgimiento de un problema colectivo. Sin embargo en este escenario surgieron las primeras inquietudes y tentativas, alimentadas por la lectura del texto *La complejidad de la cooperación* (Axelrod, 2004), en particular el artículo sobre difusión de la cultura

- Dado que *microcultura del aula* es extremadamente diversa se necesitaba un factor unificante para la resolución colectiva de problemas y un tamaño mínimo de la cultura compartida, expresada en las prácticas culturales dentro del aula, como la publicación constante, la lectura y el comentario entre pares y las mediaciones tecnológicas de programación.

- El factor unificante no se ubicaba en la *microcultura del aula* sino en la macrocultura compartida, en ese sentido, un uso de los computadores profundamente influenciado por lo multimedial era un referente común y una aplicación de programación con énfasis en lo multimedial y la facilidad de uso fue escogida para todos: Squeak

- Una buena parte del trabajo extra clase tuvo que ver con ubicar literatura apropiada que ayudara en el aprendizaje de Squeak. Los tiempos asociados a preparar la experiencia y hacer una metalectura de ella fueron un factor limitánte de la pertenencia a la microcomunidad de práctica por parte del docente, lo cual no permitía encausar todo el proceso, *desde adentro*.

- Se empezaron a ver los primeros indicios de trabajo colaborativo y la respuesta favorable grupal por el uso de una herramienta común, de las características antes mencionadas, fue generalizada, si bien no llegamos a la solución colectiva de problemas.

- Se modificó el instrumento de evaluación para que diera cuenta de modo explícito de los valores que se querían propagar en la microcultura (pudo estar relacionado con el artículo de surgimiento de normas sociales de Axelrod en *La complejidad de la cooperación*, el instrumento de evaluación explicitaba la norma).

Segundo momento: coherencia y competencia, pero no modularidad

Con la experiencia del anterior semestre se eligió Squeak desde el comienzo como un entorno de programación orientada a objetos (POO), fácil, multimedial, que hace uso de *eToys*, un sistema de programación visual de arrastrar y soltar. Se trataba primero de adquierir el tamaño mínimo de la cultura compartida para solucionar problemas, así que después de cubrir los temas del comienzo del curso, se inició la parte introductoria a la programación, estableciendo un mínimo de actividades comunes que, si bien podrían tener ligeras variaciones de estudiante a estudiante usaban los mismos conceptos fundamentales de la programación orientada a objetos (instanciación, agregación, polimorfismo y encapsulación) y el manejo de la interface.

Las actividades de consolidación del tamaño mínimo de la cultura compartida fueron dos: la creación de un libro interactivo, usando las facilidades del *Bookmorph* de Squeak y un juego interactivo usando los *eToys*. Luego de esto se procedió a resolver un problema colectivo, crear un juego, que a su vez era la resolución de un problema colectivo: sacar a un conjunto de bichos virtuales de un laberinto, mediante la asignación de funcionalidades individuales, diferentes y complementarias para cada uno de ellos (existe un popular juego de los 80's para consolas de juegos llamado *Lemmings*, con varios clones como *Pingus*, el nuestro era un clon más en ese sentido; pueden verser capturas de pantalla en el anexo 5). Esto permitía tener dos niveles de lectura sobre la resolución del problema: a nivel de los estudiantes y a nivel del juego mismo.

Cada estudiante debía encargarse de programar una funcionalidad específica para el bicho y deberían poder integrarse todas en un escenario virtual donde varios ejemplares del bicho (instancias en el sentido de la programación orientada a objetos) estarían encerradas. Esta segunda experiencia mostró varias cosas referidas a la interacción entre los agentes (estudiantes, bichos virtuales) y el entorno (salón-mediaciones, laberinto):

- El espacio común de interacciones entre los agentes permitía coordinar acciones entre ellos. Cuando una funcionalidad que era pertinente a varios subproblemas emergía, esta actuaba como *atractor* (en el sentido de los sistemas dinámicos complejos). Esto se veía tanto en las páginas wiki de publicación de los estudiantes, así como en la formulación de preguntas *a otros estudiantes* durante el aula (se iba formando una *microcomunidad de discurso*[20]), y también en los códigos de mayor fun-

cionalidad de los bichos virtuales o de los entornos que habitaban. Tanto en el caso de las páginas wiki, como de la charlas en el aula y del código en los objetos de software y sus entornos los *atractores* ayudaban a organizar la actividad de resolución de problemas por parte de los agentes bien fueran otros estudiantes en el curso (agentes naturales) o bichos de software (agentes artificiales).

- Se presentaron los primeros casos de bifurcación: soluciones alternativas al mismo problema de funcionalidad. Esto mostró una primera aproximación a la modularidad en términos del "valor de opción" (fulano y megando), que precisamente tiene que ver con la posibilidad de elegir diferentes implementaciones de respuestas posibles a un mismo (sub)problema. La solución más robusta obraba como atractor en el sentido antes descrito y era elegida después de ser puesta a prueba en el aula, mediante la ejecución de su funcionalidad en el entorno para el que se había creado y de discutirse por estudiantes y profesor.

- El tamaño mínimo de la cultura compartida se puede ver en los indicadores de competencia y coherencia de los agentes individuales y a su vez estos son interpendientes de la modularidad. Como bien lo indica la literatura de sistemas multiagente (perencejo año), la coherencia es fácil de lograr o se presupone como una condición del sistema. En este caso se hizo un diseño de ambiente de aprendizaje que la embebiera mediante un proyecto común con subproyectos individuales, así como instrumentos de evaluación que poderaban y alentaban la coherencia. Una vez se tiene un conocimiento compartido suficientemente grande y si el diseño del problema lo permite, surgirán las soluciones alternativas que pueden ser intercambiadas en un entorno. En el caso de los estudiantes era fácil ver las soluciones alternativas, en el caso de los agentes de software, la coherencia se había *forzado en el diseño* (como en su correlato de aula) y la competencia y modularidad se podían ver en la cantidad de mensajes diferentes (métodos en POO) que estaban en condiciones de entender en común y la forma en que dichos mensajes se implementaban.

- Habíamos alcanzado un límite respecto a la modularidad y esto se debía a que el

20. Nos referimos a este concepto desde Fish (1980), citado por Brown y otros en su escrito *Conocimiento especializado distribuído en el aula* (Compilado en Salomon 1993), en la que «los participantes son iniciados en los rituales del discurso y la actividad académicos y, más específicamente, científicos».

diseño de los *eToys* deliberadamente colocaba un límite sintáctico a las posibilidades expresivas con el lenguaje, de modo que no se colocaran complejidades extras, para las que no estaban preparados, a los niños, quienes eras los usuarios objetivo del mismo. Sin embargo, esto restringía las posibilidades de intercambiar pequeñas partes del código que implementaban funcionalidades específicas, teníamos que compartir objetos completos con todos sus métodos o usar funcionalidades extendidas no provistas por *eToys* que hacían complicado el manejo de la interface en este punto y, si bien contabamos con ayuda de miembros de la comunidad de *Squeakers* internacional que implementaron algunos métodos de más bajo nivel por nosotros, pasar de la programación a través de mosaicos arrastrando y soltando, a un código sencillo sintácticamente, pero que no habíamos leído previamente era muy difícil en ese punto[21] y por tanto no podíamos modificarlo, adaptarlo o subdividirlo para garantizar la funcionalidad completa de las partes del proyecto.

Tercer momento: coherencia, compentencia y modularidad

Para el tercer semestre la experiencia se reinterpretó y rediseño a la luz de los aprendizjes de los dos semestres anteriores. Para no enfrentar inconvenientes límite sintáctico de *eToys* se pensó en establecer un puente entre ellos y el código en Smalltalk que está por debajo de su funcionamiento, para lo cual se usó una variante creada en Squeak, llamada Bots Inc (Ducasse, 2005), que está especialmente concebida para la enseñanza de los conceptos de programación, al mismo tiempo que presenta un correlato gráfico en los algoritmos, desde la idea de programación de robots virtuales de software que pueden dejar trazos en un entorno (en una inspiración similar a la de Logo, pero desde el entorno objetual multimedial de Squeak). Se trataba de extender la funcionalidad de los robots de modo que estuviera en condiciones de crear un laberinto y salir de él, afrontando diferentes peligros (huecos) y premios (bonus). Para crear la cultura compartida (aumentar los índices de coherencia y compentencia), no sólo se replicó el diseño del entorno educativo en términos de los instrumentos de evaluación, las metodologías, mediaciones y contenidos (eduwiki, Squeak, *Bookmorphs*, *eToys*, etc), sino que se extendió para incluir nuevas mediaciones y contenidos, en particular el libro y el entorno de Bots Inc, lo que permitía

21. Esto se debía a que el código se había visto informálmente, pero que no había sido estudiado con detenimiento previamente al estar oculto, hasta el final del semestre, bajo la interface gráfica.

acceder a componentes más modulares del código e intercambiarlos sin mayor problema. Además de los atractores y bifurcaciones presentados en el caso anterior se podrían resaltar estos momentos del correlato educativo del modelo:

- Se pasa de el tamaño mínimo de la cultura compartido, un indicador grupal o de entorno, inspirado en la terminología de Axelrod, a coherencia y competencia, un indicadores asociados a los agentes individuales. La cultura compartida del sistema tiene un correlato en la coherencia y comptencia de los agentes.

- Se explicita la modularidad como la característica que no permtía intercambiar trozos de código cuando se alcanzó el límite sintáctico de los *eToys* y se establece una correlación entre modularidad y coherencia: los agentes estarán dispuestos a asumir roles activos en la solución del problema colectivo, si los trozos están bien definidos, son *suficientemente* pequeños (en alguna métrica de tamaño) y son intercambiables.

- Se configuró un espacio de aula con coherencia (los instrumentos de evaluación, las mediaciones y las dinámicas valoraban y alentaban mucho el trabajo conjunto), competencia (se cubrieron y evaluaron contenidos iguales hasta cierta parte del semestre) y modularidad (se diseñó un problema colectivo con subproblemas interdependientes en un entorno dónde el código fuera fácilmente intercambiable).

- Se empezaron a ver los primeros fenómenos emergentes. La pregunta por la emergencia en el aula fue una de las más difíciles en el correlato educativo. ¿Qué era aquello que, en términos del correlato educativo del modelo de resolución colectiva de problemas, estaba en el sistema, pero no en sus componentes individuales?. Para apreciar el fenómeno nos adscribimos a la noción de aula que proponen Brown y otros:

  «Teóricamente concebimos el aula como compuesta por zonas de desarrollo próximo (Vigotsky, 1978) a través de las cuales los participantes pueden desplazarse por diferentes rutas y a diferentes velocidades (Brown y Reeve, 1987). Una zona de desarrollo próximo puede incluir a personas, adultos y niños, con diferentes grados de conocimiento especializado, pero puede abarcatar también artefactos tales como libros, videos, láminas murales, equipos científicos y un contexto infor-

mático destinado a apoyar el aprendizaje intencional (Campione, Brown y Jay, 1992; Scardamalia y Bereister, 1991). Una zona de desarrollo próximo es la región de actividad que los alumnos pueden recorrer con ayuda proveniente de aun contexto de apoyo que incluya a personas pero no se limita a ellas (Vigotsky, 1978). Define la distancia entre los niveles reales de compresión y los que pueden alcanzarse en colaboración con personas o con artefactos poderosos. La zona de desarrollo próxima encarna un concepto de disposición a aprender que subraya niveles superiores de competencia»

La propiedad emergente está referida precisamente a la resolución del problema, dado que esta es una capacidad del sistema, que no se encuentra individualmente en ninguno de los agentes. Por supuesto, decir que, si hay competencia y coherencia, entre todos solucionamos un problema mejor que cada uno, es una premisa básica de la cual parte la resolución de problemas en sistemas multiagente, como se vió en la sección respectiva. El aporte del modelo es asociar a estas dos características de los agentes, una del problema, que es la de modularidad. Pero el énfasis del fenómeno emergencia, en términos de correlato educativo del modelo, no está allí, sino en el hecho de que, como un todo, el sistema se hace más competente (de modos no lineales) con el aumento de las competencias individuales y a pesar de que esta competencia grupal afecta positivamente la competencia individual, una vez los agentes/estudiantes son considerados en soledad (como mostraron las evaluaciones del curso) la competencia decrece considerablemente. La emergencia se da en tanto que las zonas de desarrollo superpuestas del aula, modifican la competencia del sistema para la resolución de problemas y dejan "resagos" en las competencias individuales. La competencia grupal es un fenómeno emergente que surge del hecho de que las competencias individuales dejan huellas visibles en el entorno constituido por un problema modular.

8 Conclusiones y recomendaciones

La resolución colectiva de problemas puede ser caracterizada efectivamente, de modo general, a partir de tres propiedades: coherencia, competencia y modularidad. Las dos primeras son propiedades de los agentes, mientras que la última es una propiedad del problema (entorno). Este es, hasta donde la revisión literaria permitió apreciarlo, el primer modelo que tiene en cuenta las tres propiedades en conjunto. La coherencia y la modularidad pueden ser vistas en términos de una propiedad del sistema, entendida con el tamaño mínimo de la cultura compartida, desde el modelo cultura dentro de la perspectiva de Axelrod 2004, es decir, como un conjunto de variables que poseen unos valores. Cuando las variables tienen los mismos valores se habla de regiones del cultura y ellas implican, en términos de coherencia y competencia, intenciones y vocabularios compartidos por los agentes, los cuales a su vez se emplean y explicitan en constructos más complejos (métodos y mensajes de programación orientada a objetos, foros wiki, etc.).

Microcambios generan macrocomportamientos. En este sentido se valida lo expuesto por Schelling en su modelo de segregación, aplicándolo también a la resolución colectiva de problemas. Esto se explicita en cómo la experiencia de aula intento propagar un *ethos* y un *conocimiento* comunes en los estudiantes individualmente a partir de instrumentos de auto y heteroevaluación y como se consideraba la publicación y el uso individual de las mediaciones, a pesar de que pretendía finalmente evaluar un proyecto colectivo, sin descuidar por ello los procesos y aportes individuales y a los subproblemas. El fenómeno emergente de la resolución colectiva de problemas, que estaba por encima de las capacidades individuales de cada uno de los agentes fue una de las evidencias del macrocambio. Estos microcambios ocurren porque cada uno de los agentes incorpora las intenciones y los vocabularios como reglas permanentes de su interacción con el entorno (como en el caso de la simulación de la colonia de hormigas) o bien porque los agentes se propagan, en su interacción, las reglas entre sí (como en la simulación de la epidemia). Sería interesante ahondar más en el correlato educativo de esta conclusión desde las ideas de zonas de desarrollo próxima y su jalonamiento en la relación estudiante-estudiante y estudiante-entorno y la metáfora empleada del aula como un espacio donde se superponen múltiples zonas de desarrollo.

Entre más cerca se está a hallar la *solución ideal* más tiempo tardan los agentes en realizar aportes a dicha solución. Este fue uno de los fenómenos emergentes más interesantes al correr la simulación computacional del modelo repetidas ocasiones. Debido a que los agentes ya han aportado los saberes en su recorrido por el espacio de búsqueda, ahora se trata de pasar por aquel lugar específico del espacio donde aún el aporte no se ha hecho, sin embargo mucha de la exploración ocurre en donde ya se ha aportado la solución, que es un lugar más grande que el lugar donde el aporte específico falta. Efectivamente en el correlato de aula, una vez habíamos integrado gran parte del prototipo del proyecto colectivo, la depuración de errores específicos y la resolución de codependencias no previstas tomó mucho más tiempo que la integración original y de hecho no se alcanzó a realizar del todo, debido a los costos que tiene lograr el tamaño mínimo de la cultura compartida. Es importante, por esto, una vez logradas coherencia y competencia en el sistema, ofrecer correlatos curriculares que le permitan al sistema continuar existiendo y funcionando para aportes más específicos a (sub)problemas puntuales.

Narduzzo y Rossi (2003) muestran como la modularidad es una característica arquitectónica que, en el caso específico del desarrollo de software, mejora las posibilidades de que hayan contrubuciones. En términos más generales del modelo acá presentado, podríamos decir que la modularidad del problema afecta positivamente la coherencia de los agentes. Esto se vió claramente en el correlato de aula, cuando, una vez alcanzado el límite sintáctico provisto por los *eToys* fue muy difícil realizar la integración de los subproblemas individuales que hacían parte del problema general, mientras que usando *Bots Inc* la posibilidad de deconstrucción e integración aumento y al final se logró resolver el problema colectivo planteado.

El aumento progresivo de la competencia se ve favorecido por metáforas deconstruibles y continuas. Los conceptos básicos de la programación orientada a objetos (herecia, encapsulación, polimorfismo y agregación) se abordaron varias veces a través de mediaciones y actividades distintas pero *continuas* y complementarias: al principio se ofreció la construcción de un *Bookmorph* interactivo (para establecer continuidad con actividades previas como el uso de presentaciónes en *PowerPoint* u *OpenImpress*), luego los mismos elementos básicos se emplearon en la construcción de un juego interactivo para enfatizar los componentes de programación y encontrar el límige de los *eToy* y posteriormente se

pasó *Bots Inc* donde se mantenía un correlato gráfico de los conceptos algoritmicos y de programación orientada a objetos, pero se tenía acceso a constructos sintácticos mucho más simples, de bajo nivel y poderosos. Lo deconstruible de la metáfora tiene que ver con los aumentos de la competencia del agente cognitivo y gracias a dichos aumentos se incrementa la modularidad y el valor de opción dentro de las soluciones al problema.

Las codependencias no previstas son parte de un problema de descomposición (Narduzzo y Rossi 2003) y es bueno tener funciones integradoras que ayuden a resolverlos (por ejemplo el "macro" método que invocaba los métodos particulares en el correlato de aula). Alguna de las funcionalidades de los módulos-métodos que resolvían problemas específicos al ser integradas en el problema general mostraban codependencias no previstas entre los subproblemas que afectaban la integración y en este caso lo que se hizo fue relegar comportamientos de los métodos específicos e integrar las partes que resolvían las codependencias al método general. En resumen, las codependencias no previstas en las soluciones a los subproblemas pueden ser resueltas en un relegando funcionalidad conflictiva de la solución a los subproblemas e integrándola en un mecanismo de comunicación entre ellos.

Los mecanismos externos de representación y la compentencia del sistema son interdependientes. En la medida en que se incrementa la competencia del sistema se hacen más complejos, densos y frecuentemente usados. En el caso de Eduwiki se pudo evidenciar cómo se pasaba de páginas personales sin comentarios, a páginas con comentarios superficiales y luego a la constitución de un foro virtual donde se compartían avances frecuentes del proyecto común, respuestas generales a los inconvenientes de integración y unos grupos ayudaban a otros en la integración, del mismo modo se empezaron a usar sistemas de representación externos como *Internet Archive* que permitía compartir todo el estado del sistema *Bots Inc* cuando la integración lo requirío.

Algunas actividades en las que el sistema educativo ha favorecido usualmente el trabajo individual pueden ser reinterpretadas y beneficiadas a la luz de la cognición distribuida, por ejemplo la resolución de problemas en general y en particular la programación de computadores (que se ha visto como una actividad que realiza y aprender un programador en "solitario" frente a su máquina). En ese sentido este trabajo se suscribe a lo dicho por Dillebourg (1996) en cuanto a que las principales funciones del entorno de aprendizaje (entre ellas el el diágnostico, la tutoria y la explicación) son principalmente

distribuidas y aporta modelos y mediaciones que explicitan cómo lo es en particular la solución de problemas. También se comparte la visión de que esto tiene consecuencias en el diseño de los sistemas educativos y de hecho el correlato de aula del modelo mostró un posible diseño que se puede emplear a fin de evidenciar y favorecer la resolución colectiva de problemas.

Diseños de aula y evaluación pueden favorecer la coherencia y la competencia aumentando la conciencia del sistema sobre sí mismo (emergencia fuerte). El instrumento de evaluación utilizado durante las experiencias de aula también evolucionó y tuvo un caracter diagnóstico y alentador de competencia individual y coherencia (la evaluación dinámica favorece la coherencia y la competencia de los agentes). Lo primero porque si un saber ya evaluado mejoraba, la nota en el instrumento lo reflejaba y lo segundo porque un elemento permanente de evaluación eran los comentarios hechos a las páginas de otros y la integración del trabajo individual en el proyecto común de modo que se alentaba a leerse entre sí y a aportar soluciones conjuntas o a otros.

Pueden haber correlatos curriculares que tenga que ver con saberes comunes que luego toman subproblemas distintos y afectan la experticia de quienes los tienen o usan/modifican la preexistente. En este sentido se pueden repensar los currículos de modo tal que los cursos ofrezcan contenidos curriculares similares (que configuran el tamaño mínimo de la cultura compartida), pero también niveles de especialización al interior de ellos, de modo que se puedan generar variaciones sobre esos saberes comunes aplicados a subproblemas específicos o que permitan usar otros saberes en los subproblemas. Esto ayuda a la configuración de una comunidad de discurso y práctica pues genera equilibrio entre los saberes comunes y los complementarios.

Posteriores estudios sobre este mismo tema podrían mejorar el modelo computacional y las aplicaciones y diálogos con el correlato educativo, en particular en lo referido a diseño de redes de aprendices que solucionen problemas colectivamente y a las mediaciones que las favorezcan, enfatizando dos aspectos: el paso del agente individual a la agencia colectiva y la validación no sólo funcional, sino también comportamental del modelo o las modificaciones al mismo a la luz de ser más cercano a dichas validaciones y explicitar aún más los alcances y las limitaciones del modelo.

9 Bibliografía

- A Guide for Writing Research Papers based on Styles Recommended by The American Psychological Association
- Axelrod, Robert (2004). La complejidad de la cooperación. Fondo de Cultura Económico. Argentina.
- Baldwin, Carliss Y. y Clark, Kim B. (2003). The Architecture of Cooperation: How Code Architecture Mitigates Free Riding in the Open Source Development Model. Disponible en Internet:
  `http://opensource.mit.edu/papers/baldwinclark.pdf`
- Bellifemine, F.; Caire, G.; Poggi, A. y Rimassa, G. (sin año). JADE A White Paper. Disponible en Internet: `http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf`
- Bordini, R y Hübner, J. (2005). Jason A Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net. Disponible en Internet:
  `http://jason.sourceforget.net/jason.pdf`
- Brow, Matthew. Cartesian Cognitivism and Its Discontents. (2004). Disponible en Internet. `http://thm.askee.net/articles/cartesian-cogn.pdf`
- Cavallo, David y el Future of Learning group at the MIT Media Lab (2004). Models of growth – towards fundamental change in learning environments. En BT Technology Journal. Vol 22 No 4 . October 2004.
- Cederman, Lars- Erik y Gulyas, Laszlo. (2001). Tutorial Sequence for RePast: An Iterated Prisoner's Dilemma Game. Disponible en Internet:
  `http://www.econ.iastate.edu/tesfatsi/RepastTutorial.IPD.pdf`
- Collier, Nick (sin fecha). RePast: An Extensible Framework for Agent Simulation. Social Science Research Computing. University of Chicago Chicago, IL. Disponible en Internet:
  `http://www.econ.iastate.edu/tesfatsi/RepastTutorial.Collier.pdf`
- Cost Scott y otros (sin fecha). TKQML: A Scripting Tool for Building Agents. Disponible en Internet: `http://citeseer.ist.psu.edu/cost97tkqml.html`

- Cunningham, Ward; Jeffries, Ron; Kessler, Robert R. Williams, Laurie (2000) Strengthening the Case for Pair Programming. En IEEE Software July/August 2000.

- Cycorp (2002). OpenCyc Tutorial. Disponible en Internet:
  `http://opencyc.org/doc/tut/?expand_all=1`

- Dastani, Mehdi y otros. (sin año). Programming Agent Deliberation. An Approach Illustrated Using the 3APL Language. Disponible en Internet:
  `www.cs.uu.nl/3apl/publication/agentdeliberation.pdf`

- Dastani, Mehdi (2004). 3APL Platform User Guide. Disponible en Internet:
  `http://www.cs.uu.nl/3apl/download/java/userguide.pdf`

- Dillenbourg, Pierre. (1995). From Mutual Diagnosis to Collaboration Engines: Some technical implications of distributed cognition on the desing on interactive learning environments. Edited transcrip of an invited talk at the World Conference on Artificial Intelligence in Education. Disponible en Internet.
  `http://tecfa.unige.ch/tecfa/publicat/dil-papers-2/Dil.7.2.9.pdf`

- Ducasse, Stéphane (2005). Squeak: Learn Programming with Robots. Apress California.

- Evans, Phillip y Wolf, Bob (2005). Collaboration Rules. En: Hardvard Business Review.

- Freeh, Vincent; Madey, Gregroy y Tynan, Renee. (Sin Fecha). Modeling the Free/Open Source Software Community: A Quantitative Investigation. Disponible en Internet:
  `http://www.nd.edu/~oss/Papers/BookChapter.pdf`

- Galoppini, Roberto y Garzarelli Giampaolo (2003). Capability Coordination in Modular Organization: Voluntary FS/OSS Production and the Case of Debian GNU/Linux. Disponible en Internet:
  `http://econpapers.repec.org/paper/wpawuwpio/0312005.htm`

- Guillaume, Blondel y Ludwig, David (2006). Modèle de ségrégation de Schelling Rapport de projet Intelligence Artificielle distribuée. Universidad de Caen.

- Harasim, Linda y otros. (1998). Redes de Aprendizaje. Gedisa. España.
- Heylen y otros. (2004). Research Proposal for a "Geconcerteerde Onderzoeksactie" Modelling the Emergence and Evolution of Distributed Cognition. Disponible en Internet: http://pespmc1.vub.ac.be/Papers/GOA-project.pdf
- Hutchins, E. (1995). Cognition in the Wild. M.I.T. Press. Cambridge, Massachusetts.
- Josephson, Susan (2000). Thinking like a Machine. Knowledge Systems approach to Artificial Intelligence. Disponible en Internet: http://www.cse.ohio-state.edu/lair/Main/TLM.html
- Kerckhove, Derrick. (1999). Inteligencias en conexión. Gedisa. España.
- Kumar, Ashwani. (2005). i-Seek: An Intelligent System for Eliciting and Explaining Knowledge. Disponible en Internet: http://agents.media.mit.edu/projects/iseek/ashwani-ms.pdf
- Kumar, Ashwani; Sundararajan, Sharad C. y Lieberman Henry. (sin fecha) Common Sense Investing: Bridging the Gap Between Expert and Novice. Disponible en Internet:

  http://agents.media.mit.edu/projects/investing/lb366-kumar.pdf
- Laird, John; Lehman, Jill Fain y Rosenbloom, Paul (2006) A gentle introduction to soar, An architecture for human cognition: 2006 Update. Disponible en Internet:

  http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf
- LinGO Matrix Project (2004). Grammar Engineering, Introduction, overview, HPSG basics. Disponible en Internet: http://courses.washington.edu/ling471/0329.pdf
- Liu, Hugo y Lieberman, Henry (sin fecha) Metafor: Visualizing Stories as Code. Disponible en Internet: http://web.media.mit.edu/~hugo/publications/papers/IUI2005-metafor.pdf

- Liu, Hugo; Lieberman, Henry y Selker, Ted (2003). Visualizing the Affective Structure of a Text Document. Proceedings of the Conference on Human Factors in Computing Systems, CHI 2003, April 5-10, 2003, Ft. Lauderdale, FL, USA. ACM 2003, ISBN 1-58113-637-4, pp. 740-741.

- Liu, Hugo; Lieberman, Henry y Selker, Ted. (2002). GOOSE: A Goal-Oriented Search Engine With Common- sense. In De Bra, Brusilovsky, Conejo (Eds.): Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002, Malaga, Spain, May 29-31, 2002, Proceedings. Lecture Notes in Computer Science 2347 Springer 2002, ISBN 3-540-43737-1, pp. 253-263.

- Liu, Hugo y Sing, Push (2004) ConceptNet – a practical commonsense reasoning tool-kit. En: BT Technology Journal . Vol 22 No 4 . Octubre 2004.

- Liu, Hugo y Sing, Push. (sin fecha). Commonsense Reasoning in and over Natural Language. Disponible en Internet: `http://web.media.mit.edu/~push/CommonsenseInOverNL.pdf`

- Macal, Charlas M. y North, Michael J. (2005) Tutorial on agent-based modelling and simulation. Center for Complex Adaptive Agent Systems Simulation (CAS2) Decision & Information Sciences Division Argonne National Laboratory Argonne, IL 60439, U.S.A.

- Maxwell, John W. (2006). Tracing the Dynabook: A study of Technocultural Transformations. University of British Columbia. Disponible en Internet:

  `http://thinkubator.ccsp.sfu.ca/Dynabook/dissertation`

- Minsky, Marvin (1986). Society of Mind. Simon & Schuster. Sidney, Australia.

- Minsky, Marvin y Sing, Push (2003). An Architecture for Combining Ways to Think. En Proceedings of the International Conference on Knowledge Intensive Multi-Agent Systems. Cambridge, MA. 2003.

- Musa, Rami; Kulas, Andrea; Anguilete, Yoan y Scheidegger, Madleina (sin fecha). GloBuddy, a Tool for Travelers. Disponible en Internet:

  `http://www.media.mit.edu/~lieber/Teaching/Common-Sense-Course/Projects/GloBuddy/GloBuddy.pdf`

- Narduzzo, Alessandro y Rossi, Alessandro (2003). Modularity in Action: GNU/Linux and Free/Open Source Software Development Model Unleashed. Universidad de Bologna y Universidad de Trento. Disponible en Internet:
  `http://rock.cs.unitn.it/file/NarduzzoRossiFOSS2003.pdf`
- Pan, Denis (2004). L'émergen
- Pfeifer, Rolf y Scheier Christian. (1999). Understanding Intelligence. M.I.T. Press. Cambridge, Massachusetts.
- Salomon, Gavriel (compilador) (1993). Cogniciones distribuidas. Consideraciones psicológicas y educativas. Amorrortu Editores. Buenos Aires.
- Sicilia, Miguel Angel (sin fecha). Sobre la concepción de conocimiento en el proyecto OpenCyc. Disponible en Internet:
  `http://serbal.pntic.mec.es/~cmunoz11/sicilia36.pdf`
- Sing, Push (2003). Examining the Society of Mind. Disponible en Internet:
  `http://web.media.mit.edu/~push/ExaminingSOM.html`
- Leigh Tesfatsion. (sin fecha). Agent-Oriented Programming. Department of Economics, Iowa State University. Disponible en Internet: `http://www.econ.iastate.edu/tesfatsi/AOPRepast.pdf`
- Weiss, Gerhard (1999). Multiagent Systems A modern Approach to Distributed Artificial Intelligence. M.I.T. Press. Cambridge, Massachusetts.
- Williams, Laurie (2000). Strengthening the Case for Pair Programming. IEEE Software Magazine. EEUU. Disponible en Internet:
  `http://www.computer.org/software/so2000/pdf/s4019.pdf`
- Yoshiki, Ohshima (sin fecha). Kedama: A massively-parallel tile-scriptable particle system. Disponible en Internet: `http://www.is.titech.ac.jp/~ohshima/squeak/kedama/`
- Yoshiki, Ohshima (sin fecha). Fun with Kedama. Disponible en Internet:
  `http://www.squeakland.org/fun_projects/kedama/kedma_getstart.htm`

10 Anexos

10.1 Anexo 1: Los agentes de razonamiento práctico desde el modelo de creencias, deseos e intensiones

Si el razonamiento práctico implica la deliberación, entonces debemos sopesar cuantos recursos computacionales vamos a gastar en ella. La deliberación permanente inhibiría la acción, pero la acción no deliberada nos coloca del lado de la reacción. ¿Qué factores se deben tener en cuenta entonces, para sopesar la deliberación y la acción?.

El modelo de creencias, deseos e intensiones (que abreviaremos CDI, para una sigla más latina), establece una importancia capital a estas últimas como una forma de llegar a la acción. Una vez ocurre la deliberación, se tiene la intensión de realizar algo y se establece un compromiso con dicha intensión (que implican varios deseos)[22], que modifica nuestras creencias. Siguiendo a Wooldridge (pág 69):

- Las intensiones guían el razonamiento orientado a fines y medios: una vez se adquiere una intensión, el razonamiento se basa en los medios que pueden alcanzar un fin.
- Las intensiones persisten: Cuando se tiene una intensión, ella no se abandona sin un conjunto de buenas razones y si se falla en un intento, se reintenta antes de abandonarla
- Las intensiones restringen futuras deliberaciones: Una vez se tiene una intensión, opciones inconsistentes con ella no serán consideradas.
- Las intensiones influencian las creencias sobre las cuales el razonamiento práctico futuro está basado: Si se tiene una intensión, entonces los planes para el futuro se basarán en que, en algún punto, dicha intensión se alcanzó y dicho logró hará parte de las creencias hacia el futuro.

Las creencias en este modelo se referirán a la información que el agente tiene a través de sus percepciones del estado del entorno, lo cual implica que el agente tenga también un estado, donde dicha información, además de la de las intenciones y deseos, está almacenada; podríamos decir que se refiere al lugar donde las representaciones del estado mental

22. El deseo puede ser visto también como el inicio de la intensión, pero en este contexto asumiremos que se refiere al compromiso adquirido con la intensión, después de que ella a surgido como fruto del proceso de deliberación.

del agente se encuentran. En palabras de Bordini y Hübner (pág ??) las creencias capturan actitudes informativas, los deseos, actitudes motivacionales y las intenciones actitudes deliverativas de los agentes.

La deliberación del agente se modelará mediante dos funciones. La primera que genera opciones a partir de las crencias (Cre) y las intensiones (Int) y las convierte en o deseos[23] (Des):

$$\text{opciones: } \wp(\text{Cre}) \times \wp(\text{Int}) \rightarrow \wp(\text{Des})$$

y la segunda que filtra las opciones obtenidas de la primera función a partir de las creencias, los deseos y las intensiones y las convierte de nuevo en intensiones, es decir que este proceso pone las intensiones previas en contexto y las mantiene o transforma en el futuro. Formalmente se tendría:

$$\text{filtro: } \wp(\text{Des}) \times \wp(\text{Cre}) \times \wp(\text{Int}) \rightarrow \wp(\text{Int})$$

en cuanto a las creencias ellas se actualizan usando las creencias previas y las percepciones:

$$\text{acr: } \wp(\text{Cre}) \times \text{Per} \rightarrow \wp(\text{Cre})$$

El ciclo de trabajo de un agente esencialmente consta de cuatro etapas (Wooldridge, 2001):

- Observar el mundo y actualizar sus creencias.
- Deliberar sobre las acciones a tomar determinando la intensión a lograr.
- Usa el razonamiento por medios y fines para encontrar un plan a ejecutar
- Ejecuta el plan.

Una descripción detallada de este proceso puede ser encontrada en el algoritmo del anexo 1 y en la explicación de un intérprete del proceso para el lenguaje formal AgentSpeak(L) en el anexo 3.

10.2 Anexo 2: Algoritmo del Ciclo de control de un Sistema de Razonamiento Práctico

Adaptado de Wooldridge (2005) pág 76

$C \leftarrow C_0$ `# actualiza las creencias`

23. Las opciones son entonces deseos.

```
I ← I_0          # actualiza las intensiones
mientras verdadero haga:
   obtenga siguiente percepto p mediante las funcion sensor
   # actualice las creencias de acuerdo al nuevo percepto
   C ← acr(C, p)
   # actualice los deseos con opciones surgidas de las creencias e
intensiones
   D ← opciones (C, I)
   # actualice intensiones. Aplicar filtro sobre creencias deseos e
intensiones
   I ← filtrar (C, D, I)
   # Elija un plan de acuerdo a las creencias, intensiones y acciones
   π ← plan(C, I, Ac)
   mientras no (vacio(π) o exitoso(I, B) o imposible (I, B) ) haga:
      α ← cabecera(π)
      ejecute(α)
      π ← cola(π)
      obtenga siguiente percepto usando funcion sensor(...)
      C ← acr(C, p)
      si reconsidera(I, C) entonces
         D ← opciones (C, I)
         I ← filtrar (C, D, I)
      fin-si
      si no sonoro(π, I, B) entonces
         π ← plan(C, I, Ac)
      fin-si
   fin-mientras
fin-mientras
```

10.3 Anexo 3: Modelo BDI implementado por el intérprete de AgentSpeak(L)

Traducción libre del autor de Bordini y Hübner (2005) pág 8.

Daremos algunos detalles más sobre el funcionamiento de un intérprete AgentSpeak(L), que está cláramente diagramado en la siguiente figura (reproducida de Machado y Bordini, citado por Bordini y Hübner (2005)). La descripción pictórica de tal intérprete, dada en la figura, facilita grándemente la comprensión del intérprete para AgentSpeak(L) propuesta por Rao. En la figura, los conjuntos (de creencias, eventos, planes e intensiones) están representados como rectángulos. Los diamantes representan la selección (de uno de los elementos de un conjunto). Los círculos representan alguna clase de procesamiento involcurado en la interpretación de los programas de AgentSpeak(L).

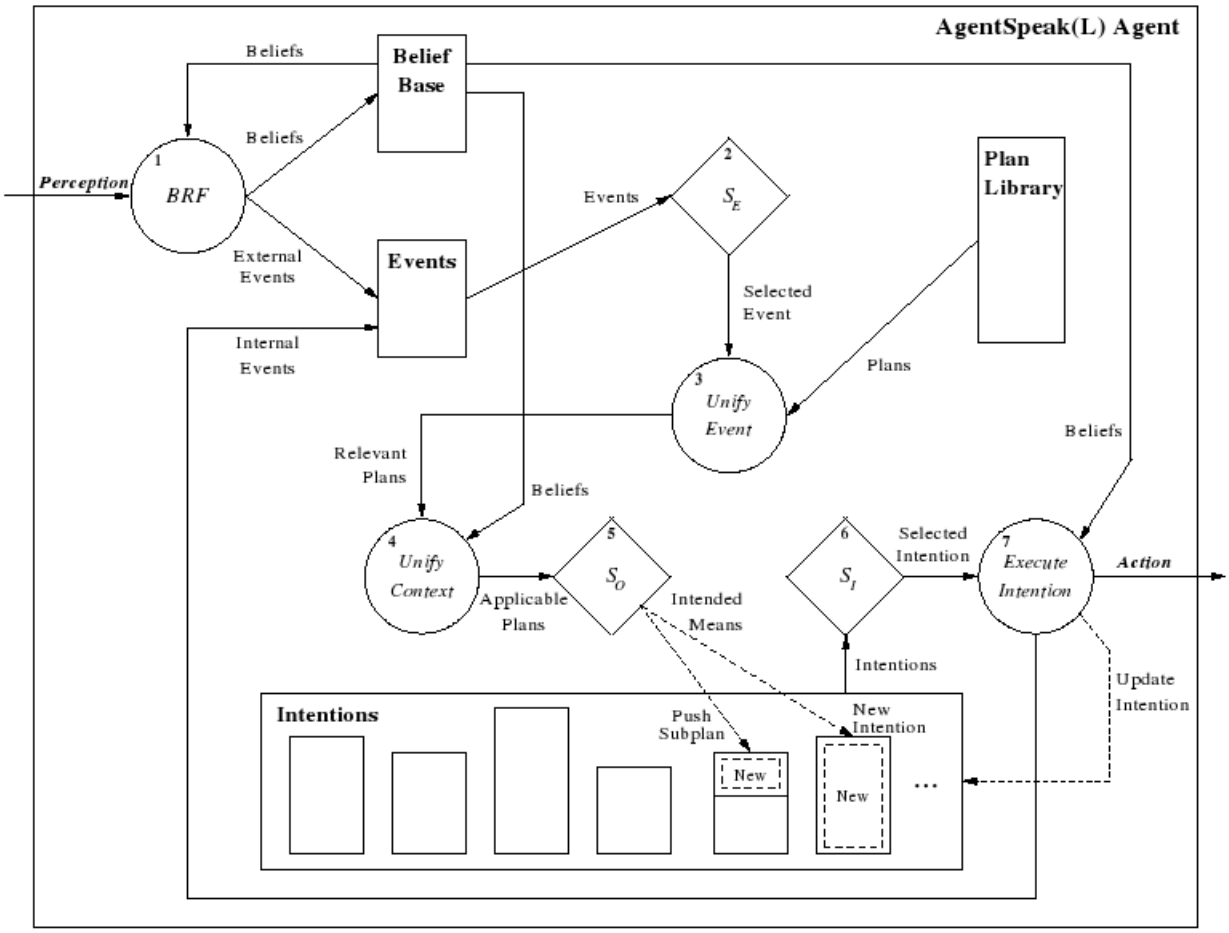

Figura 7. El ciclo de ejecución del intérprete AgentSpeak(L) para programación de (multi)agentes en sistemas de razonamiento práctico (Tomado de Bordini y Hübner (2005)).

En cada ciclo de interpretación de un programa agente, AgentSpeak(L) actualiza una lista de eventos, que pueden ser generados desde la percepción del entorno, o desde la ejecución de intensiones, (cuando los sub-objetivos son especificados en el cuerpo de los

planes). Se asume que las creencias son actualizadas desde la percepción y cuando hay cambios en las creencias de los agentes, esto implica la insersión de un evento en el conjunto de los eventos. Esta función de revisión de creencias no es parte del intérprete de AgentSpeak(L), sino más bien un componente necesario de la arquitectura de agentes.

Después de que $\mathcal{S}_\mathcal{E}$ ha seleccionado un evento, AgentSpeak(L) tiene que unificar tal evento con los eventos disparadores en la cabecera de los planes. Esto genera un conjunto de *todos los planes relevantes*. Chequeando si la parte contextual de los planes en aquel conjunto se deduce de las creencias del agente, AgentSpeak(L) determina un conjunto de *planes aplicables* (planes que de hecho pueden ser usados en aquel momento para manejar los eventos escogidos). Entonces $\mathcal{S}_\mathcal{O}$ escoje un único plan aplicable de ese conjunto, que se convierte en los *medios pretendidos* para manejar aquel evento, y o bien empuja el plan a la cima de una intensión existente (si el evento era uno interno), o crea una nueva intesión en el conjunto de intensiones (si el evento era externo, e.j. generado por la percepción del entorno).

Todo lo que queda por hacerse en este estado es seleccionar una simple intensión para ser ejecutada en el ciclo. La función $\mathcal{S}_\mathcal{I}$ selecciona una de las intensiones del agente (es decir, una de las pilas independientes de planes parcialmente instanciados en el conjunto de intensiones). En la cima de esa intensión hay un plan, y la fórmula al comienzo de su cuerpo es tomada para su ejecución. Esto implica que o bien una acción básica es realizada por el agente en su entorno, o un evento interno es generado (en caso de que la fórmula seleccionada sea un objetivo a lograr), o una prueba es realizada (lo cual significa que el conjunto de creencias tiene que ser probado).

Si la intensión es realizar una acción básica o un objetivo de prueba, el conjunto de intensiones necesita ser actualizado. En el caso de que un objetivo de prueba, la base de creencias será buscada por un átomo de creencia que se unifique con el predicado en el objetivo de prueba. Si esa búsqueda es exitosa, una creación posterior de una instancia de variables ocurrirá en el plan parcialmente instanciado que contiene el objetivo de prueba (y el objetivo de prueba en sí mismo es removido de la intensión de la cual fue tomado). En el caso donde una acción básica es seleccionada, la actualización necesaria del conjunto de intensiones es simplemente remover esa acción de la intensión (el intérprete informa al componente arquitectónico responsable por los efectores del agente qué acción es requerida). Cuando toda la formulación en el cuerpo del plan ha sido removida (es decir, ha sido ejecutado), todo el plan es removido de la ejecución, y así el objetivo de logro que lo genero (si ese era el caso). Esto termina el ciclo de ejecución y AgentSpeak(L) empieza

todo de nuevo, verificando el estado del entorno despúes de que los agentes han actuado sobre él, generando los eventos relevantes y así sucesivamente.

Se concluirá este anexo con el formalismo de AgentSpeak(L), adaptado de Bordini y Hübner, que define qué es un agente desde el modelo de creencias, deseos e intensiones y presenta detalladamente la definición de un plan (aquello que en nuestro enfoque, al ser ejecutado resuelve el problema).

- Un agente *ag* es un listado de creencias *cs* y de planes *ps*:

$$ag ::= \text{cs}\ \ ps$$

- Las creencias son con un listado de al menos una o más fórmulas atómicas (at) :

$$bs ::= at_1 \ldots at_n \quad (n > 0)$$

- Una fórmula atómica *at* es un predicado P sobre un conjunto de términos t_i expresados en lógica de primer orden:

$$at ::= P(t_1, \ldots, t_n) \quad (n > 0)$$

- Los planes *ps* son un conjuntos de al menos un plan p_i :

$$ps ::= p_1 \ldots p_n \quad (n > 1)$$

- Un plan p se define como:

$$p \ ::= \ ed : ct \leftarrow h$$

  donde

  - *Lo que hace que el plan se inicie*: *ed* es un evento disparador.
  - *Lo que hace que el plan se mantenga*: *ct* es un contexto, referido también como la cabecera del plan, que es lo que se debe cumplir mientras el plan se ejecuta. El contexto debe ser una consecuencia de las creencias del agente para considerarse aplicable.
  - *Las instrucciones del plan, propiamente dichas*: h, también referido como el cuerpo que es una secuencia de acciones, objetivos o actualizaciones de creencias

- Un evento disparador *ed* puede ser la adición de una creencia ($+at$) ó la remosión de una creencia (-at) ó la adición de un objetivo ($+o$) ó la remosión de un objetivo (-o).

$$ed \ ::= \ +at \mid -at \mid +o \mid -o$$

- El contexto ct puede ser o bien verdadero (no se evalua ninguna condición) o buen el resulado de la evaluación simultánea (operador `&`) de una serie de predicados sobre términos atómicos:

$$\begin{aligned} ct &::= \texttt{verdadero} \mid l_1 \& ... \& l_n \ (n \geqslant 1) \\ l &::= at \mid \texttt{no}(at) \end{aligned}$$

- El cuerpo del plan h puede ser o bien verdadero (el plan no ejecuta nada en particular) o bien una secuencia de funciones f_i, con $i \geqslant 1$. Dichas funciones son acciones sobre términos u objetivos o actualizaciones de creencias.

$$\begin{aligned} h &::= \texttt{verdadero} \mid f_1 ; ... ; f_n \ (n \geqslant 1) \\ f &::= A(t_1, ..., t_n) \mid o \mid a \end{aligned}$$

- Los objetivos pueden ser de logro (mantenga un estado del entorno, representado por $!at$) o de prueba (cambien un estado del entorno, representado por $?at$)

$$o ::= !at \mid ?at$$

- Las actualizaciones de las creencias consisten en la adición o remoción de términos atómicos

$$a ::= +at \mid -at$$

10.4 Anexo 4: Tutorial introductorio a la arquitecturay el uso de Kedama

Traducción libre del autor de Yoshiki, Ohshima (sin fecha). Fun with Kedama. Disponible en Internet: `http://www.squeakland.org/fun_projects/kedama/kedma_getstart.htm`

10.4.1 Empezando con Kedama

Kedama está hecho de varias partes. En la parte inferior está el mundo Kedama en sí mismo, el cual aparece como una gran caja negra. Encima de estos está una grilla invisible de cuadrados. Cada cuadrado es llamado un patch (parche), y la grilla entera es controlada por variables parche que puedes programar. Los parches sólo son encontrados en Kedama y pueden ser pensados como de un papel de gráficos con 100 x 100 celdas (10.000 en total). A cada celda puede ser asignado un valor numérico. Moviéndose a través de eso s parches en el mundo Kedama están los rebaños de tortugas, que son grupos de tortugas que actuan en la misma forma. Puedes decidir cuántos diferentes rebaños hay, establecer

cuántas tortugas hay en cada rebaño, escoger un color de rebaño, y escribir guiones que todas las tortugas en cada rebaño seguirán. Estas tortugas, moviendose a lo largo del mundo Kedama, pueden también leer el valor de cada parche y responder a él. Finalmente, hay tortugas individuales, o partículas, que comparten en mismo nombre que su rebaño. Por ejemplo, el primer rebaño que creaste será "tortuga1, así que cada tortuga en este rebaño seguirá las instrucciones a la tortuga1. A las tortugas se les da las instrucciones como rebaño, no como tortugas individuales.

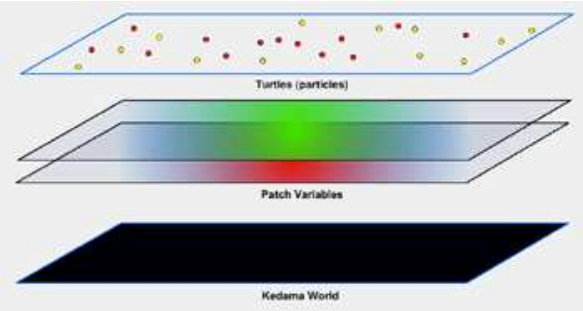

Figura 8. Arquitectura de capas del mundo Kedama

Para encontrar Kedama en Squeak: Ve a las Provisiones y arrastra el catálogo de objetos. Deberías ver un botón etiquetado "Kedama".

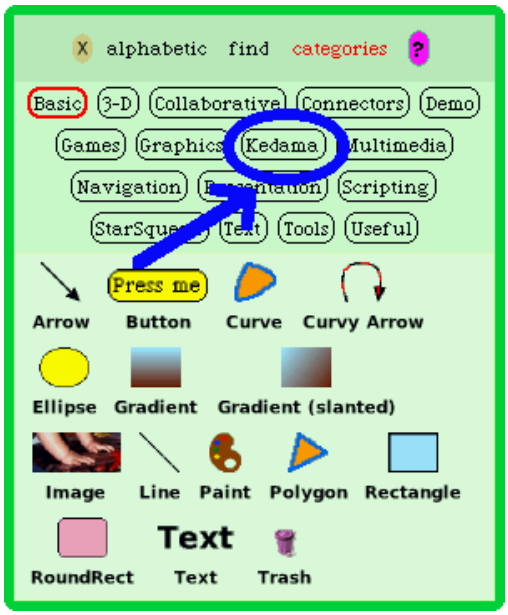

Figura 9.

Continua y haz click sobre el botón Kedama para revelar una pequeña imagen a escala del mundo Kedama. Cerca a este hay un pequeño icono etiquetado "KedamaWorld-

Bundle". Este botón también puede ser útil y regresaremos a el prontamente.

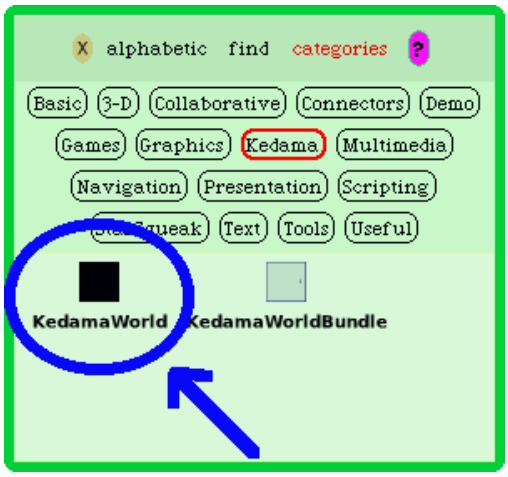

Figura 10.

Arrastra el mundo kedama al mundo etoy, y estás listo para arrancar! Saca un visor del mundo Kedama y encontrarás una nueva categoría llamada "kedama", que contiene los comandos que pueden ser usados dentro del sistema.

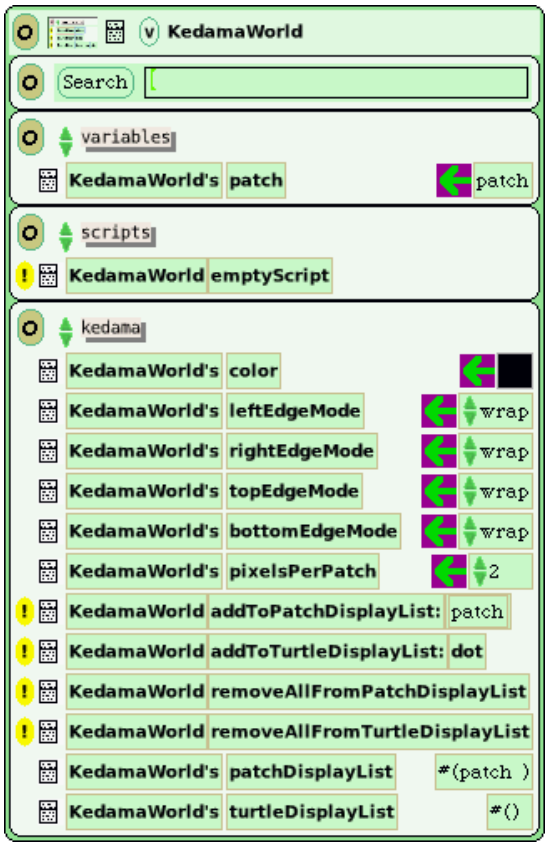

Figura 11.

Aquellos que están usando comptuadores con resoluciones más altas podrían querer jugar con el comando de Kedama “pixelsPerPatch”, localizado en el mismo menú. Acá, puedes cambiar el número de pixels en cada parche para hacer las tortugas en el mundo Kedama más claras para ti.

10.4.2 De los átomos a las tortugas

Vamos a crear un guión simple. Vamos a crear muchas torutas para representar átomos en un sistema y hacerlos mover.

Pasos:

1. Crear un nuevo rebaño de tortugas.

2. Establecer el número de tortugas en ese rebaño.

3. Hacer que todas las tortugas se muevan.

Paso 1: Crear un nuevo rebaño de tortugas.

Para crear un nuevo rebaño de tortugas, haz click sobre el icono encontrado en la parte superior del visor y selecciona “add a new breed of turtle” (agregar nuevo rebaño de tortugas).

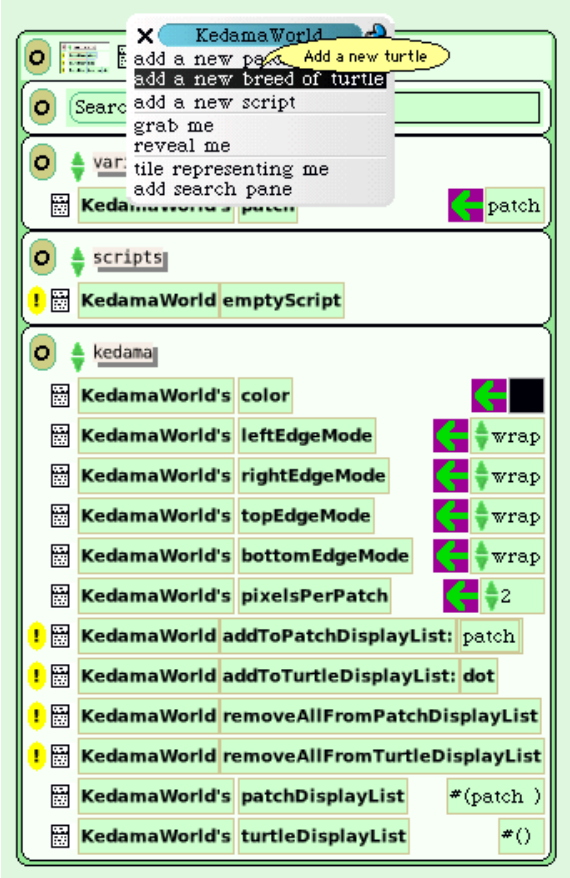

Figura 12.

Espera un momento y obtendrás una caja coloreada en la posición de tu cursor. Esta es la representación de un rebaño de tortugas en el mundo. Sirve como un ejemplar o un modelo, para todas las tortugas que crees de ese rebaño. Un pequeño punto coloreado (una tortuga) también aparecerá en el mundo Kedama. El primer rebaño de tortugas siempre es rojo. Si creas más de un rebaño de tortugas, los siguientes colores son asignados aleatoriamente.

Figura 13.

Paso 2: Configurando el número de tortugas. Para obtener el número de tortugas puedes hacer una de dos cosas, dependiendo de lo que quieres que tus tortugas hagan.

1. Si quieres que todas las tortugas se comporten de la misma forma y sigan el mismo guión, entonces querrás agregar tortugas del mismo rebaño como el primero. Para mostrar el visor para tu rebaño de tortugas, apunta el cursor sobre el ejemplar, o el cuadrado rojo que representa el rebaño, y deberías ver el familiar halo de manejadores. Abre un visor y escoje la categoría "kedama turtle breed".

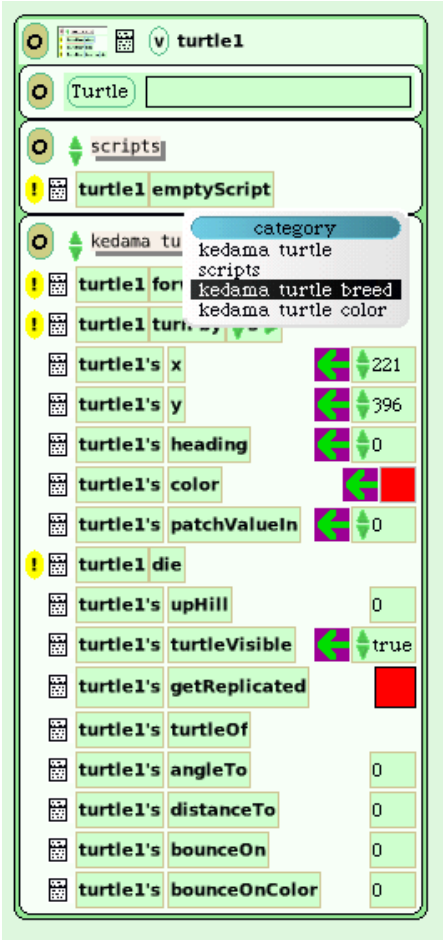

Figura 14.

¿Ves el comando que dice "turtle1 turtleCount" y el número asignado?

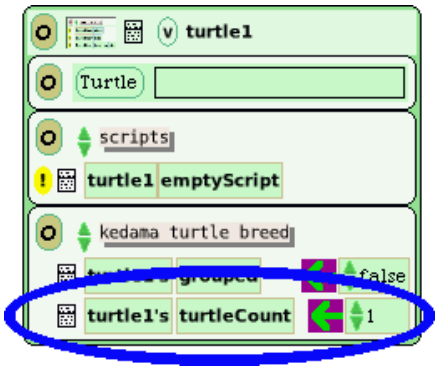

Figura 15.

avanza y cambia aquel número a la cantidad de tortugas que quieras. Ellas deberían aparecer en el mundo Kedama tan pronto como tecleas "Enter" y serán del mismo color que tu primer tortuga. Ten presente que incluso aunque pueden haber muchas tortugas, todas ellas comparten el mismo nombre puesto que ellas son del mismo rebaño - tortuga 1.

2. Si quieres que las nuevas tortugas se comporten de forma diferente y sigan un guión diferente, querrás agregar un nuevo rebaño de tortuga. Haz esto del mismo modo que agregaste el primer rebaño de tortugasy verás un punto coloreado diferente y un ejemplar aparecer.

Para nuestra simulación, queremos que todas las tortugas se comporten en la misma forma - como átomos - así que usaremos el primer método. Cuando hayas finalizado, deberás ver el mundo Kedama con muchas tortugas rojas en él, todas del mismo rebaño.

Paso 3: Hacer que las tortugas se muevan:

Hagamos que nuestro rebaño de tortugas se mueva. Es muy similar a hacer otros objetos en Squeak moverse, así que probablemente ya eres un experto en esto! Si no está ya abierto, anda y abre un visor para la tortuga y escoje la categoría “kedama turtle”. Deberás ver una lista de comandos familiares incluido el “turtle1 forward by”.

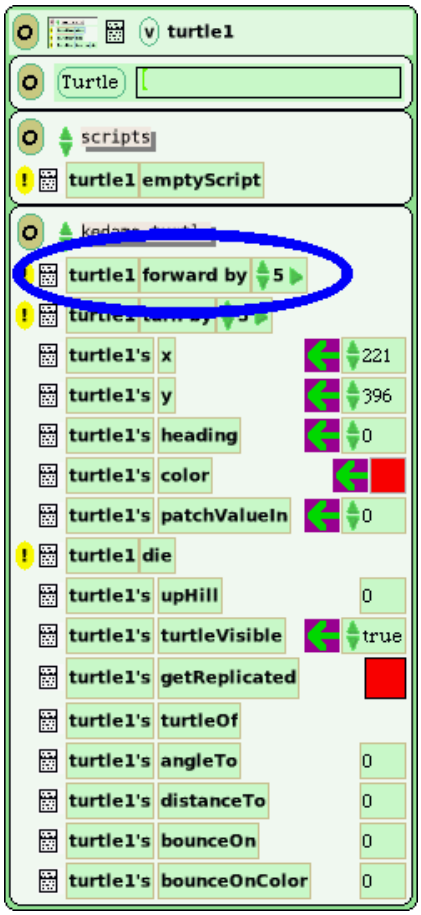

Figura 16.

Arrastra este mosaico para obtener un nuevo guión. Entra cuánto quieres que las tortugas se muevan cada vez, pon tu guión a ejecutarse y mira a las tortugas avanzar. Felicitaciones - has usado ahora Kedama para hacer una simulación atómica.

10.4.3 Atomos que rebotan

¿Viste cómo las tortugas que se movian en nuestra recien creada simulación parecían dejar en mundo Kedama y entrar de nuevo por el otro lado? Esto es llamado "*wrapping*", (envolvimiento) puesto que las tortugas parecen evolverse alrededor de un eje al eje opuesto. Si quieres que las tortugas reboten de los lados, sin embargo, puedes hacer esto haciendo un cambio en el menú "kedama". Optén el menú "kedama" haciendo click en el mundo Kedama y abriendo un visor, o haciendo click en el mundo kedama en la solapa. Deberías ver diferentes comandos, incluyendo la palabra "EdgeMode". Hay cuatro de ellos corres-

pondientes a los cuatro vértices, y están actualmente configurados en "*wrap*" (ajustado, envuelto). Para hacer que los atómos reboten en los vértices, puedes cambiar su estatus a "*bouncing*" (rebotando).

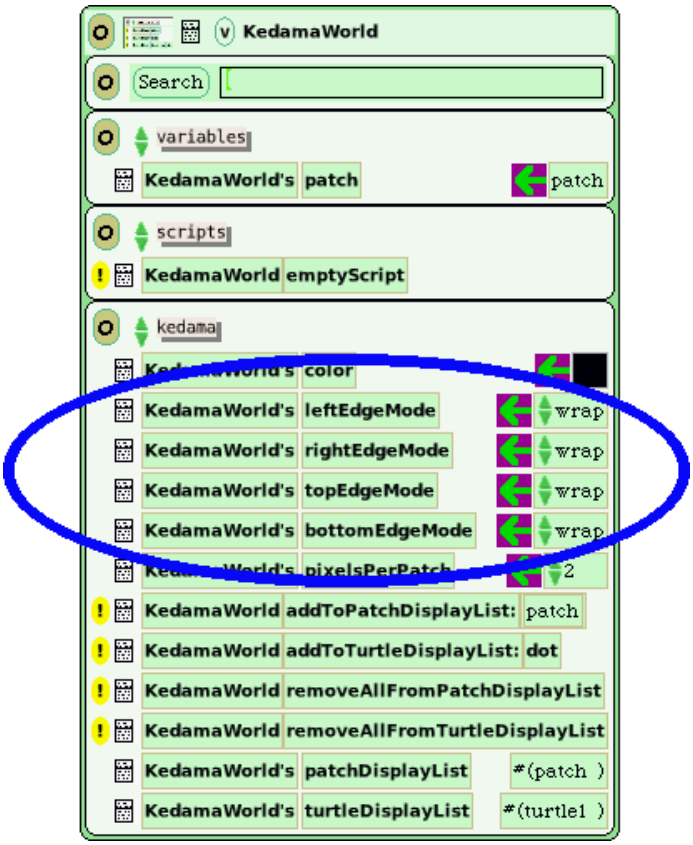

Figura 17.

Ejecuta tu script de nuevo y mira que pasa!

También puedes controlar la velocidad a la cual los átomos se mueven. ¿Ves el ícono del reloj en la parte superior de tu guión?. Si mantienes presionado el mouse sobre él, verás el siguiente menú:

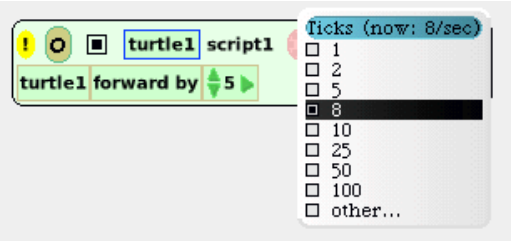

Figura 18.

Esto te permite configurar cuantos *ticks*, o ciclos, ocurren cada segundo. Anda y configura la taza de ciclos a diferentes valores para observar qué pasa.

10.4.4 Diversión con Color

Podemos también adicionar algunos interesantes efectos visuales con color usando los parches en el mundo Kedama. Primero detén tu escript mientras hacemos los cambios. Desde el visor de la tortuga, arrastra el mosaico "patchValueIn" y suéltado en tu guión.

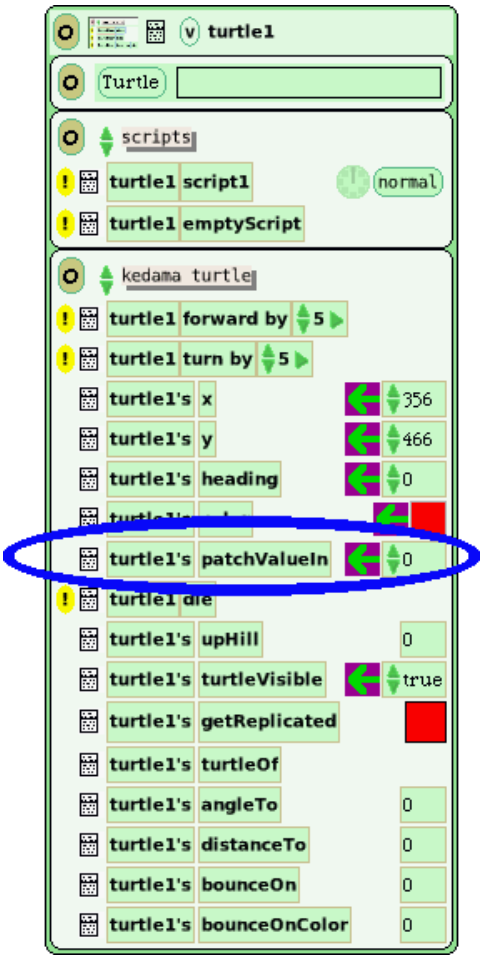

Figura 19.

Esto te permite asignar un valor numérico a cada parche. Los números que escojas también influencian los colores de los parches, en la medida en que los números son más grandes, incrementan el brillo del color del parche. Por ejemplo, cambiando el valor del

mosaico a 10 pondría un valor pálido en el parche donde la tortuga actualmente reside. Por ahora, cambia el valor del argumento del mosaico a 40. Entonces, presione el pequeño triángulo ascendente a la izquieda de "patchValueIn" para cambiarlo por "patchValueIn patch increase by".

Adelante, corre tu guión. Nota que el color en el pequeño rectángulo se hace azul en la medida en que las tortugas dejan detrás un rastro de puntos coloreados.

Si quieres que tus tortugas dejen rastros de diferente color, puedes cambiarlo usando el visor del parche. Para ver el visor para los parches abre el visor Kedama y seleccione el icono próximo a la variable "Kedama's patch".

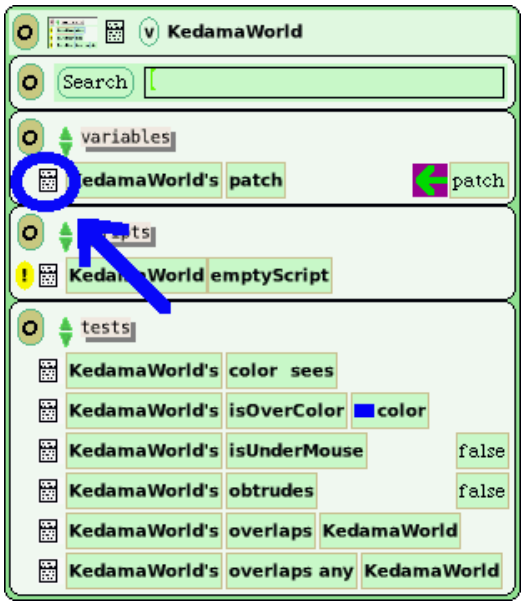

Figura 20.

Escoge "grab morph", y una pequeña caja que luce como un mundo mini-Kedama aparecerá. Esta es una representación de un parche en el mundo Kedama. Colocala en tu pantalla, además del mundo Kedama. Ahora, haz click en el morph del parche y abre su visor. Deberías ver un menú de categoría "kedama" ya abierto. Entonces, haz click en la caja coloreada en el mosaico u201cpatchu2019s coloru201d y selecciona el que quieras.

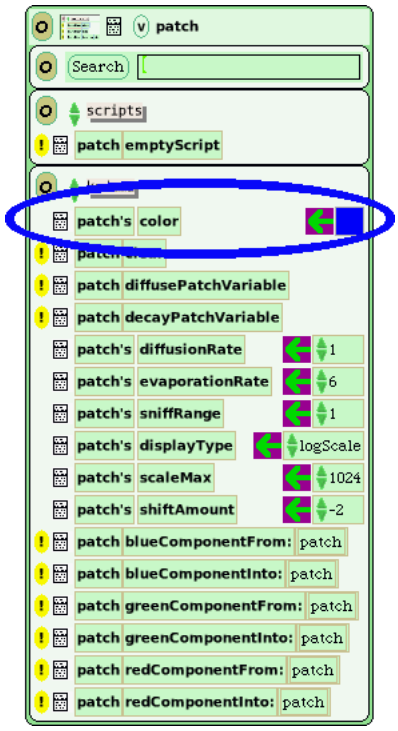

Figura 21.

Pausa tu script para unos pequeños cambios más. Los efectos acá serán más fáciles de ver si hay más pocas tortugas, así que cambia el `turtleCount` a "20". Luego, dile a tus tortugas que se muevan hacia adelante en 2 unidades.

Luego, encuentra el mosaico “scaleMax” en el visor del parche y configura su valor a "200".

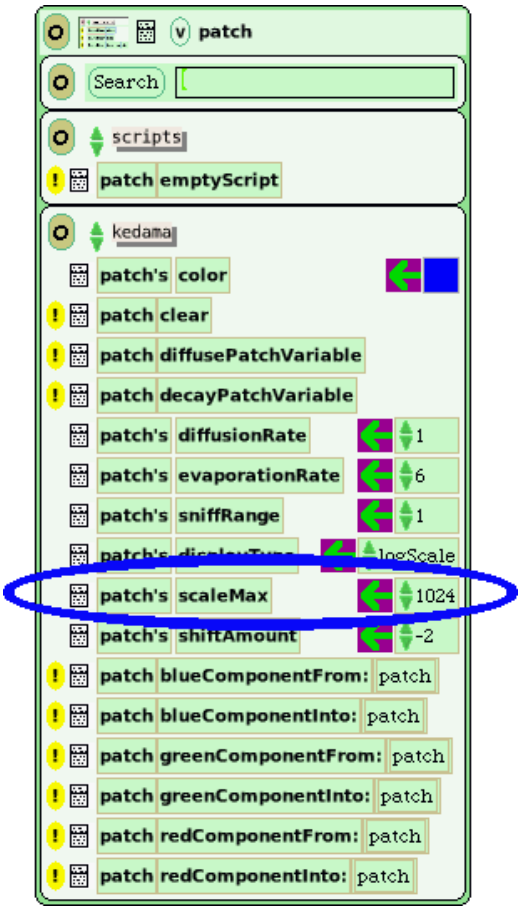

Figura 22.

Haciendo esto, configuras el límite superior de saturación de color para el parche. Si quieres, puedes cambiar este valor y observar qué pasa al color contra el fondo negro de Kedama.

Finalmente arrastra "patch diffusePatchVariable" y suéltalo en el scriptor (el lugar donde se crean los guiones/scripts). Esto adicionará el efecto visual de los rastros de las tortugas difundiéndose o exparciéndose, detrás de ellas.

Está casi listo para partir. Para preparar el mundo para tu nuevo guión, presiona la marca de exclamación amarrilla a la izquierda de "patch clear" para borrar los rastros de puntos azules de los parches en el mundo Kedama.

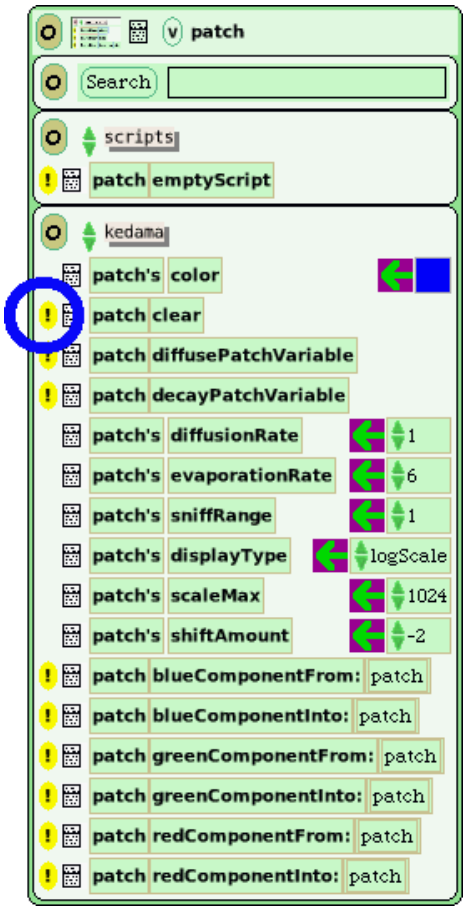

Figura 23.

Ejecuta tu guión! Verá a las tortugas dejando un "humo" en la medida en que se mueven.

Las fases en un scriptor serán ejecutadas en una manera que peude ser llamada "sincronización line-wise". Esto significa que una línea será ejecutada después de que todo lo relacionado con la línea anterior se halla completado. Por ejemplo, en éste guión, "patch diffusePatchVariable" es ejecutado después de qe cada tortuta en el rebaño tortuga 1 ha compeltado la acción "patchValueIn:". Usando esta idea, puedes cambiar cómo los rasgos de las tortugas aparecen cambiando donde pones el “diffusePatch Variable” en el guión. Mira cuidadosamente – ¿puedes ver la diferencia?

Una cosa más: ¿recuerdas la miniatura etiquetada “KedamaWorldBundle”?. Si fueras a arrastras ese icono en el mundo etoy, lo que verías sería esos tres componentes ya atados juntos, listos para que lo uses: un mundo Kedama, un morph parche, y un rebaño de tortugas.

10.5 Anexo 5: Correlato educativo del modelo

En este anexo se encuentran las capturas de pantalla de los proyectos realizados indagando por el correlato educativo del modelo de resolución colectiva de problemas en sistemas multiagente y algunos extractos de Eduwiki, donde se evidenció cómo creció la competencia del sistema.

Figura 24. Pingus, un clon de un juego de arcade de los 80's llamado Lemmings, en el cual se soluciona un problema de modo colectivo. Agentes relativamente sencillos, los pingüinos, deben resolver el problema de salir de un escenario con diferentes peligros y dificultades (abismos, barreras, etc.) para lo cual usan conocimiento especializado que puede ser asignado a los agentes (taladrar, usar paracaidas, volar, ser bomba, entre otros), en ese sentido el problema es modular (descomponible en pequeñar partes fuertemente ligadas al interior y vagamente ligadas entre sí). Si el porcentaje de pingüinos salvados satisface o supera un mínimo prestablecido el considera se considera resuelto.

Figura 25. Uno de nuestros primeros intentos recursivos. En el aula resolveríamos colectivamente un problema en el cual, a su vez, programamos un juego que resuelve un problema colectivamente. El clon de pingus tenía pingüinos que interactuaban con el entorno y, para definir su comportamiento, diferentes estudiantes programaban diferentes guiones interacciones con el entorno mediante el envío de mensajes (taladrar, caer, detener a otro, etc.) y debían integrarse en el juego. Enfrentamos el problema de que el entorno de programación (*eToys*) no era suficientemente modular y las soluciones a los subproblemas no podían ser fácilmente intercambiadas/deconstruídas/integradas.

El jugador humano, una vez terminado el programa, debía seleccionar el mensaje que se quería enviar al pingüino para que ayudara a los demás a salir, evitando los peligros, usando para ellos los botones azul, rojo y verde de la esquina superior izquierda.

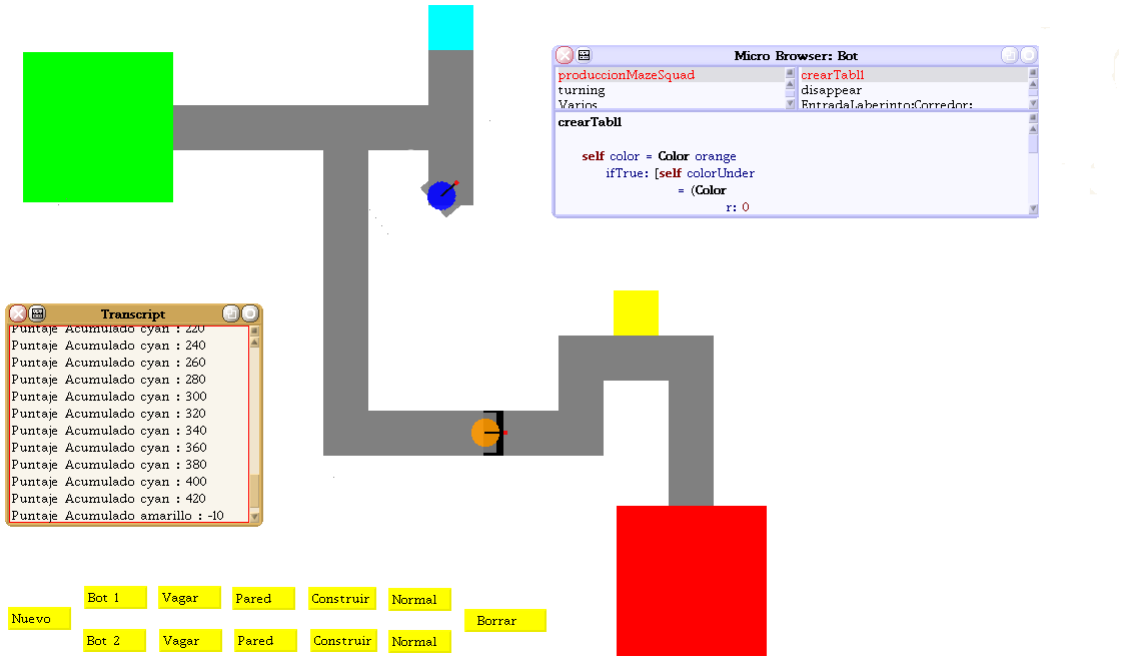

Figura 26. *Maze Squad* (el escuadrón del laberinto): la última iteración del proyecto en el tercer semestre del correlato de aula. Inspirado en el trabajo de los dos semestres anteriores, pero programado en el entorno Bots Inc, que, a diferencia de *eToys*, permite el acceso ilimitado a todos los métodos y clases del lenguaje subyacente Smalltalk, al mismo tiempo que provee una metáfora sencilla y con retroalimentación gráfica del funcionamiento de los algoritmos a través del comportamiento de los robots en el entorno. Se hacen otro tipo de simplificaciones respecto a entornos profesionales de programación, como el *microworspace* y el *microbrowser* de clases (en azul en la gráfica), que le permiten a los estudiantes leer, entender, extender e intercambiar funcionalidades de los robots para la resolución de un problema colectivo.

Los botones amarillos corresponden a las diferentes funcionalidades habilitadas durante el juego, para que el jugador humano las selecciones. Los cuadrados rojo y el verde grandes son la entrada y salida del laberinto, respectivamente, mientras que los amarillo y aguamarina son *bonus* y *antibonus* de puntaje. La funcionalidad de «construir puente» ha sido asignada a uno de los robots, lo que se denota con su color naranja y se puede ver cómo avanza en la medida en que reduce un espacio negro (abismo) frente a sí. El robot azul vaga aleatoriamente en una parte del laberinto.

El foro del proyecto *MazeSquead*

La comunidad de discurso que consolidamos con motivo del proyecto *Maze Squad* no sólo se explicitó en las prácticas y dinámicas presenciales durantes el aula, sino que tuvo también su espacio en un foro virtual. Se pueden aprenciar varias interveciones de estudiantes que dan cuenta de la resolución colectiva de problemas, así como del anuncio de la

resolución de su parte como de los problemas de codependencias no previstas en el momento de integración. La mayoría de los aportes fueron maduros, centrados y referidas, con excepción de uno que otro reclamo injustificady y reivindicación personal.

A continuación la transcripción del foro web del proyecto (los nombres en azul corresponden a enlaces web a diferentes páginas personales o del proyecto).

¿Qué tal? ¿Cómo están? Yo soy Sergio y estoy trabajando en proyecto MazeSquad2 creando el laberinto, pero tengo una duda que no sé si ya la hayan podido solucionar o si la han tenido en cuenta: ¿cómo es el método para que el robot quede en primer plano siempre? Porque siempre que crea los rectángulos queda debajo de ellos. Espero me puedan ayudar. Gracias. – SergioAndresRivera

Aunque no lo hemos intentado hacer, sabemos que existe el problema. Offray nos dijo que investigáramos sobre `picUp`, depronto esa *pista* ayude. – SDaniel

Qué tal Sergio Daniel, si, ya en clase hoy Offray nos dio como una primera visión acerca de esa solución pues se metió a un canal de chat y ahi vió. Esperemos y podamos solucionar el problema que se nos presentó acá, en la ultima parte. Sin embargo si pueden solucionarlo primero espero que lo puedan documentar. Gracias. – SergioAndresRivera

---

- JoseRamirez: En vista del problema que han planteado, mirando en el browser de clases me encontre que existe una solucion a este problema, la cual consiste en implementar el metodo "comeToFront" de esta forma siempre el robot estara por encima del rectangulo, lo único que hay que tener en cuenta que el trazo del robot ese si queda debajo del rectangulo.

```
| cu box pica |
pica:=Bot new.
box := Rectangle center: World center extent: 200@200.
cu := RectangleMorph new.
cu bounds: box.
cu color: Color green.
```

```
cu setBorderWidth: 1 borderColor: cu color.
cu openInWorld.
pica comeToFront
```

Muchísimas Gracias por solucionar el problema. Gran aporte. – SDaniel

Lo mismo que SergiO, MUCHAS GRACIAS POR EL APORTE – SergioAndresRivera

Gracias Jose. Bastante útil.

Nuevas versiones e integración de discusión en un sólo lugar

Hola a todos,

Como me imagino que estarán desesperados por camellar en el proyecto y se habrán dado cuenta en sus intentos preliminares por integrar el laberinto a sus pruebas respectivas que requerimos de variables para cada uno de los rectángulos que lo conforman, he integrado, rápidamente, las ideas de Aura respecto a listar individualmente las variables al código del método de construcción del laberinto y he subido las últimas versiones tanto de las categorías del método como del laberinto usa las variables.

Algunas observaciones generales:

- Las pruebas de los métodos y los métodos mismos deben estar en lugares diferentes del código. No integren todo en un mismo lugar. Vean como el código que construye el laberinto y los métodos que lo construyen están separados. Así deben funcionar los otros métodos y sus respectivas pruebas.
- Como ahora cuentan con variables independientes para cada rectángulo, usenlas para probar sus métodos. Por ejemplo si el robot debe vagabundear de la entrada al primer corredor usen las variables `in` y `c1` para probar el método. Si debe caer en un hueco, usen las variables del corredor que le antecede y las de hueco mismo, de modo análogo para construir puente. El método que nos presentó Aura, que dados dos rectángulos le ayuda a pasar del uno al otro es clave, pero requerimos que publique los archivos st. Ya le dije personalmente y en alguna de los comentarios de la página. Estaremos a la espera de dicha publicación.

- Cuando invoquen la construcción del laberinto de pruebas, el entorno les preguntará reiterativamente por variables que han sido declaradas en el Bot Workspace, pero que aparecen no usadas. Respondan a estas preguntas diciendo que no quieren que se borren las variables.

Veré sus adelantos el jueves.

– offray 2007-04-30 23:53:22

- Hola Todos, Gracias a los cambios que realizo Offray en el codigo de *Contruccion-Laberinto*, ahora es posible almacenar los rectangulos del laberinto en variables, en un *arreglo*, o lo que es mejor aun una *coleccion*, pues a diferencia del *arreglo* la *coleccion* permite agregar rectangulos si un limite predeterminado, manteniendo el orden en que fueron agregados los elementos. Los metodos en *ConstruccionLaberinto* por ahora se encuentran en la version que offray actualizo, lo unico que cambié fue el script en el que se prueba la contrucción del laberinto. Asi que para probarlo solamente descargen la ultima version del script de prueba.

– DiegoFlorez 2007-05-01 00:20:40

- Hola a todos,

  He estado mirando la mejora de Diego y creo que usar una colección ordenada resuelve muchos de los inconvenientes para el manejo de elementos del laberinto. La última parte del código de pruebas del laberinto, tiene una orden que recorre cada uno de los elementos del mismo por el robot que lo crea. Les recomiendo que cambien este tiempo de 1 milisegundo a 500, con lo cual podrán ver cómo ocurre esto. – offray 2007-05-01 14:33:52

He visto las modificaciones y creo que ya todo tiene como un sentido, y algo de facilidad para todos. Sin embargo no se si haya un problema con el script que crea el laberinto, y es que a la hora de formar una "T" en el script que el grupo de construccion del laberinto (MazeSquad2) hicimos hay una superposicion de rectangulos, no se si esto afecte todos los procesos o si habria que cambiar y modificar minusiosamente los `jump` para que unicamente crear rectangulos y saltara a un punto determinado sin tener que crear un rectangulo, sobre otro rectangulo.

Otra cosa que sugiere mi atencion es que los rectangulos que van en la coleccion son los de incio, salida y crear corredor pero los de hueco y bonus no estan, entonces nose como se haria para los que estan trabajando en estos temas –SergioAndresRivera

- Hola,

  En cuanto a la superposición de rectángulos no hay problema. De hecho el método de Aura se basa en que los rectángulos están superpuestos para poder pasar de uno a otro. En lo referente a los rectángulos del hueco y el bonus, efectivamente estos deberían ir a la colección. Por favor modifiquen el código para que estos dos rectángulos queden adicionados y actualicen los enlaces respectivos. Recuerden nombrar la modificación con un consecitivo para saber que va después de otras, algo como `pruebaLaberinto4.st` – offray 2007-05-02 02:57:01

Quetal, Cuando se quiera adicionar el de hueco y bonus deben ir en la misma coleccion que los demas? o en otra coleccion distinta. He subido el codigo `PruebaLaberinto4.st` añadiendo los rectangulosde bonus y hueco a la coleccion. –SergioAndresRivera

---

- JoseRamirez: Con relación al proyento de MazeSquad2 a mi grupo nos toca hacer la parte en el que el robot vaga un poco en la entrada y despues salta al laberinto, pero teniendo en cuenta las modificaciones que se le ha realizado a la construccion del laberinto al adicionar variables, al manejar el codigo que tenemos y al tratarlo de unir al nuevo código hemos presentados nuevos problemas. Mas detallesProduccion Robots Charla.

- Hola,

  En la página de detalles he contestado asuntos específicos, sin embargo mi aclaración puntual es la siguiente: Necesitan usar las ideas del código de Aura para que un robot vagabundee de un rectángulo a otro y su método debe recibir como parámetros dos rectángulos. Vagabundea en el primero y si encuentra una intersección con el segundo, se pasa a él. – offray 2007-05-02 02:57:01

- Hola a todos:

Soy AuraCristinaGarzon me gustaria que observaran los aportes que he hecho desde hace ya un buen tiempo (casi 2 semanas) en actividades tercer cierre; respecto a construccion del laberinto, construccion hueco, contruccion puente y robot vagabundeando que se encuentran allí. Espero sea de gran utilidad para cualquiera de ustedes y pueda resolver las dudad generadas en el proceso de este proyecto esto tambien es para MazeSquad2, podrán encontrar los metodos con su respectiva implementacion y los codigos para generar todo completo. Suerte a todos.

- Hola,

  Como sólo hasta ayer publicaste los códigos, a pesar de la insistencia previa, no ha sido posible revisar muchos de los aportes. Sin embargo, respecto a lo que se publicó ayer, los índices para el laberinto son un buen aporte y sirven para tomar ciertos elementos específicos y usarlos como argumentos de otros métodos. Creo que los métodos de Diego Florez respecto a Bots Inc y el problema de desaparecer robot son bastante ilustrativos y aportan en general a los diferentes problemas que tenemos en la integración del código en el proyecto, tanto en la construcción del puente, como de caer en el hueco y de recorrido y construcción del laberinto. Por favor revísenlos y usen lo que les sirva para sus respectivos métodos. – offray 2007-05-04 10:38:29

- Un saludo cordial a todos:

Somos el grupo de JessicaCastellanos:, LauraFonseca:, FelipeCorredor:, hemos tenido un problema con la implementacion del script que realiza la funcion de que existe un robot "guia" de color rojo y otro lo sigue el robot que lo sigue cuando se encuentre en la posicion 0 con el robot guia se detenga o que rebote pero aun no le hemos logrado espero alguno de ustedes nos pueda colaborar. Gracias.

- Qué problema específico han tenido. ¿Revisaron e integraron el código de Diego que vimos en clase?. Si nos pueden mostrar documentación y código específicos es mucho más fácil ayudarles. – offray 2007-05-04 10:38:29

- Hola a todos:

Los codigos estan publicados hace ya dos semanas; nos estaban los st pero hay ilustracion del workSpace y de la ventana de metodos de cada uno de los metodos que yo he implementado y he aportado al proyecto que estamos trabajando, el dia de ayer en clase no se podia acceder desde casi ningun computador en la sala, por esta razon el profesor Offray los subio desde el computador de él. Espero estos aportes sean de gran ayuda para cualquier inquietud a MazeSquad o MazeSquad2 en espcial con el metodo llamado muevaseybusque, Hare lo mismo pero desde mi pagina personal, es decir dentro de mi pagina encontraran los st de estos metodos y tambien sus pruebas; me interesa mucho saber sus opiniones porfa comentemen que les ha parecido. Mucha suerte. AuraCristinaGarzon

Hola Aura.

Estuve viendo tus metodos, son de muchisima utilidad a la hora de recorrer el laberinto, pero aun tenemos el problema de depender de los rectangulos, osea le tenemos que decir al metodo cuales son los rectangulos que debe examinar. Seria estupendo que no dependieramos de los rectangulos sino unicamente de algo como el color, para saber si es un hueco o una salida u otro corredor. Por otra parte creo que podrias simplificarlo un poco usando n timesRepeat a la hora de controlar las iteraciones, en lugar de un while.

– DiegoFlorez 2007-05-04 19:13:08

- Hola,

  Aura, como tu misma dices los códigos _no_ estaban publicados hace dos semanas, sino las capturas de pantalla de los mismos, que son diferentes a los archivos st, y por supuesto las capturas de pantalla hacen imposible hacerle pruebas o mayores. Afortunadamente con lo que tenemos de esos códigos y, principalmente, el trabajo de Diego, es posible integrar varias de las funcionalidades al recorrido del laberinto. Esta tarde discutiremos con Diego y el resto del grupo dicha integración. – offray 2007-05-07 13:40:15

- Hola todos.

Viendo una conferencia sobre Squeak, que estaba en el blog de Offray se me ocurrió, que la solución a nuestros problemas de identificacion del laberinto, ejemplo, verificar hueco, verificar salida, verificar corredor, para que el robot ejecute sus diferentes acciones; se encuentran en el manejo de los morphs, como lo hacen los etoys, esencialmente la idea es tratar de implementar lo mismo que se hace cuando se le dice al carrito que siga un

camino y se hace un verificacion del color, no tengo ni idea de como se hace esa verificacion desde nuestro entorno, pero si a alguien se le ocurre algo creo que nos ayudaria bastante.

– DiegoFlorez 2007-05-04 19:14:58

- Hola todos.

JoseRamirez: Teniendo en cuenta que a mi grupo nos toca implementar la entrada al laberinto, hemos estado tratando de que funcione con los arreglos que se han tenido en cuenta para la construccion del laberinto, pero no funciona, si alguien puede ayudarnos en tratar de solucionar el problema pueden entrar a produccion Robots Charla, donde se encuentra el script usado junto con la imagen.Gracias.

- Hola todos.

Ya logre hacer que el robot desaparezca sin necesidad de indicarle cual es el rectangulo que forma un hueco, pero dependemos de los colores. Los Etoys hacen ese tipo de verificacion, pidiendo que un color vea a otro con el metodo **color:sees:**, pero dado que nuestro robot no es un Etoy y no conocemos exactamente su estructura gráfica aunque sabemos que es una subclase de la clase morph, no trabaja adecuadamente con el metodo **color:sees:**, ni con el metodo **touches:color:** (el cual verifica si el morph se solapa con el color indicado), pero existe un metodo que nos puede servir mucho y si funciona a la perfeccion con la clase **Bot** se llama **colorUnder** el cual retorna el color que se encuentra debajo del morph o en este caso del robot al cual lo enviemos, usando este metodo podemos verificar que color se encuentra debajo del robot y asi podemos indicarle que hacer segun el color. Pronto publicare los metodos en MazeSquad/desaparecerRobot.

Aun tenemos problemas con el trazo y otros objetos de color negro pues cuando el robot los detecta (que precisamente verifica que color esta debajo del centro del robot) hace la accion que le pidamos, pero por ejemplo en el caso de desaparecer si se encuentra con un agujero *negro* o **cualquier otro objeto con este color**, el robot desaparecera de inmediato. Para solucionar esto les sugiero que establezcamos ciertos estandares de colores; una opcion podria ser hacer ligeras variaciones a los colores que queremos y renombrarlos, por ejemplo el negro es en rgb r:0 g:0 b:0.004 (lo cual es un poco raro pues deberia ser 0 0 0, pero bueno) entonces variarlo a r:0 g:0 b:0.005 y asignarle el nombre hueco o algo asi, bueno ya no les quito mas tiempo.

– DiegoFlorez 2007-05-06 01:14:59

- JoseRamirez: Con relacion a la pregunta que el grupo de JessicaCastellanos:, LauraFonseca:, FelipeCorredor plantearon con relación a la exixtencia de un robot guia de tal manera que los otros lo sigan, una posible solución es crear una colleción de robots como los de rectangulos, de tal manero que se ha subido algunos scripts que les puede ser de ayuda segun las necesidades más especificas que tengan, estos estan en Historial produccion Robots en la última semana al final.En uno de ellos el primer robots se mueve y los otros lo siguen. Espero que sea de ayuda.
- Gracias José,

  Espero que Jessica, Laura y Felipe se inspiren en esta funcionalidad para la de ellos. Es de anotar sin embargo que los robots no deben seguir a otros, sino rebotar contra alguno de un color particular. – offray 2007-05-07 13:40:15

- Offray, quetal. Con respecto a la tarea que nos encomendó hemos tenido una serie de dificultades, a lo mejor sencillas de resolver. Abrimos el entorno de Bots luego con las teclas Alt+O abrimos el catalogo de objetos, arrastramos el boton al mundo, y abrimos su visor (estilo E-Toys) sin embargo no sabemos como "abrir el visor del robot" para poder hacer que cambie de color o que avanze, nose, funciones de ese estilo (Tal parece que el problema es bastante similar a la verificacion de color, o cosas de ese estilo que plantea diego florez mas arriba). En nuestra pagina estan las capturas de pantalla de lo que hemos hecho. – SergioAndresRivera

---

- Hola Todos, Nuevamente actualize el archivo mazeSquead.st que incluye los metodos de construir laberinto de mazeSquead2, realizado por el grupo de Sergio, adicional, ahora incluye un método llamado **crearLaberinto** que devuelve la coleccion ordenada que representa el laberinto y otro llamado **wanderingLaberinto** que recorre el laberito y se desaparece si encuentra un hueco, aunque aun tiene sus fallas si alguien quiere probarlo descargue el archivo mazeSquead.st y ejecute lo siguiente en un workSpace

```
| pixa laberinto |
pixa :=Bot new.
laberinto := pixa crearLaberinto.
pixa jumpTo: (laberinto at:5) center.
pixa wanderingLaberinto.
```

DiegoFlorez 2007-05-27 19:24:49

- José gracias de parte del grupo de FelipeCorredor, LauraFonseca, JessicaCastellanos. Con respecto a lo de los scripts que se han subido actualmente hemos trabajado pero no hemos logrado obtener el movimiento aleatorio con giro de 90 grados cuando un robot se estrella con el robot guia, nos hemos basado en un script de DiegoFlorez realizando algunas modificaciones pero no concretamos aun lo que queremos. "No sabemos que pasa o en donde estan los metodos de DiegoFlorez, parece que la pagina ya no los contiene" Si alguien sabe en donde estan o a que lugar fueron movidos agradeceria responder....Gracias

FelipeCorredor, JessicaCastellanos y LauraFonseca 2007-05-27 19:24:49

- Felipe,

Los metodos se encuentran en el siguiente link mazeSquead.st, es necesario que baje el archivo .st y lo cargue dentro de squeak.

DiegoFlorez 2007-05-27 19:24:49

- Quetal, ya con nuestro grupo hemos conseguido que un boton de Etoys pueda darle instrucciones a un robot por medio de una(s) variable(s), en nuestra pagina se muestra todo el proceso y las capturas de pantalla que muestran como un botn hace que el robot cambie de color, o que cree otro.

SergioAndresRivera 2007-05-27 19:24:49

- LO LOGRAMOS:

hola q tal mi grupo de trabajo ya realizo la mision de que el robot rebotara en el robot "guia" en la pagina esta publicado y ya se implemento en el ultimo laberinto q publicaron ahora el inconveniente es q el robot rojo inicia desde el rectangulo donde mas se demora en vagabundear por ello necesitamos cambiar la posicion inicial del robot y tambien que los dos robot vagabudeen al tiempo para que los dos caminen..

cualquier comentario que nos pueda ayudar sera bien recibido...

FelipeCorredor, JessicaCastellanos y LauraFonseca 2007-05-27 19:24:49

- Hola a todos,

En la siguiente dirección encontrarán el archivo Ready.image que está listo para descargar con todos los métodos integrados de todo el image.

http://www.archive.org/details/mazeSquadTest

Deben hacer click en el enlace que dice FTP. Reune el trabajo del grupo de Jose y Andrea, del grupo de Felipe, Sergio y Carlos, de Diego Florez y en grupo del seminario de Informática. Dado que ahora vamos a trabajar con eToys los cambios deberán ser integrados en un único image que se contenga tanto los botones de los eToys como la funcionalidad del código de los Bots. Nos queda una última semana para esto. Las instrucciones sobre cómo subir archivos a Internet Archive están el la bitácora de esta semana del grupo de Sergio, Felipe y Carlos (bueno, desde Linux, usando consola de comandos, pero hay formas gráficas de hacerlo en otros sistemas operativos).

Las futuras versiones del image deben ser enumeradas con un consecutivo según se vayan creando. Ejp: `Ready.1.image`, `Ready.2.image`, `Ready.3.image`... etc.

– offray 2007-05-11 02:04:53

Offray, pequeño detalle, falto poner la direccion ;). Y otra cosa, cuales son las funciones que tenemos que poner en los botones. Porque hay muchas, quisiera saber cuales son ya que hay muchos metodos que corresponden al mismo trabajo del grupo (por ejemplo, para crear el laberinto son como 6 métodos, entonces nos queda complicado a cada metodo asignarle un boton). Por favor colaborenos en esta organizacion para que nos rinda esta semana. Gracias

SergioAndresRivera

- Hola Sergio,

  Ya coloqué la dirección. En cuanto a los botones, será necesario uno que coloque dos robots en el laberinto y luego los ponga a vagabundear de la entrada al primer corredor (usando los métodos de Andrea y Jose) y otros que cambien el color de cualquier un robot a rojo y detenerlo, para que el otro rebote, uno por cada robot, y otro que coloque a cualquier robot de color naranja (uno por cada uno), que es el color asociado a construir un puente. Por lo pronto tenemos que usar botones redundantes, puesto que la funcionalidad que les mostré en el proyecto de StarLemmings no ha sido implementada. – offray 2007-05-11 02:26:33

Hola

Soy M.Andrea después de haber leido los aportes al foro surge una pregunta El sistema Operativo Linux como puedo acceder a la consola de comandos.

Andrea, es mejor que si quieres subir el archivo `.image` y ademas tienes windows, optes por investigar como se utilizan los metodos graficos para subir un archivo. Si tienes linux instalado, ve al menu aplicaciones, luego accesorios, y ahi terminal, o consola. (No estoy seguro si tu tienes wajig en tu pc, a lo mejor podrias tener apt-get o aptitude, nose, tendrias que mirar cual es tu gestor de paquetes por consola).

–SergioAndresRivera

- Bien, ya he comenzado a instalar los botones de E-toys en el image, ya pude hacer que un boton haga que un robot (que no se le van a modificar sus acciones) cree el laberinto. Ahora divido en dos columans que van a decir Bot 1 y Bot 2, es decir debajo de cada una pondre un boton que ejecutara dicha accion. El boton que pondré primero será el de crear un bot (este hará que cree un robot llamado "Bot 1" y lo ponga a recorrer el primer rectangulo y que pase al corredor). Sin embargo, a pesar de que hemos logrado que el robot salte al centro del rectangulo inicial, cuando intento adaptar el método de Jose `EntradaLaberinto: Corredor:` y le doy click al boton, todo el enterno se "traba" es decir, no me permite hacer nada, ni sacar los halos, ni modificar codigos, nada de eso, Lo unico que me permite es salirme :S. En la bitácora de la última semana están algunas capturas de pantallas y los problemas que tuve al realizar esto(Historial).

*Si quieren pueden bajar tambien el archivo `ready1.image` de http://www.archive.org/details/mazeSquadTest para ver los cambios, aunque no se si las variables se cambien (lean la documentaciondel historial y sabran de que les hablo). OJO CON ESO

–-SergioAndresRivera

- Hola soy JessicaCastellanos estuve revisando todos los ready.image que offray publico en la pagina, dentro de estos botones se encuentra el de crear un nuevo robot el de vagabundear pero cuando se da cilck en el boton de vagabundear solo llega hasta el final de la entrada pero vagabundea en el laberinto como tal bueno por lo menos eso fue lo que ami me paso quisiera saber ¿porq no vagabundea en todo el laberinto? e incluso cuando yo integro el metodo de "Rebotar" uno de los robots queda debajo del laberinto y otro lo crea en la salida mientras los otros dos robots cumplen la tarea de rebotar.

- MazeSquad2/ConstruccionPuente. Q mas somos el grupo de contruir puente y el metodo para crear el puente ya esta funcionando y sirve muy bien si el robot es de color naranja crea tabla y la posicion del robot cambia para que cree en forma recta las tablas y los demas robots puedan seguir sin que se caigan si un robot de otro color pasa este desaparece. PaolaMejia y Maria Fernanda Cubillos Vargas

---

- Tubimos que modificar el laberinto ya que el metodo del bonus no esta integrado con el `.image` final ya que no dejaba que el robot avanzara. El nuevo `.st` se encuentra en nuestra pagina. De igual forma, ya conseguimos que dos robots interactuaran al mismo tiempo utilizando dos work spaces (uno para cada uno), sin embargo hay que modificar los metodos sencillos e integrarlos al wandering para que sea como un método que regule el comportamiento dentro del laberinto en general.

– JuanFelipeRiverosGuevara

Aunque un poco tarde, ya hemos subido el `ready.2.image` a la pagina del internet archive, sin embargo la version que está ahi es como una "beta" ya que no esta modificado el laberinto, y faltan otros botones que estan en la estable, pero nose porque motivos, no hemos podido subir. En cuanto pueda, se los comunicaremos ;). Bueno, lo unico que falta es que metan en el wandering los métodos de construir puente y detener al otro robot. Unicamente falta eso!(Y lo del bonus claro esta :S:S).

– SergioAndresRivera

Ya he subido el `ready.2.image` a http://www.archive.org/details/mazeSquadTest

---

- HOLA A TODOS:

" Squeka tiene un defecto de ejecucion de scripts, ya que cuando se cambia la referencia del color en el codigo fuente, si hace el cambio respectivo, mientras que cuando se especifica el color PINK, no hace el cambio respectivo. Se miro el fuente de Sergio, y el de sergio efectivamente si asigna el color PINK al cuadro en cuestion".

Para la asignacion del bonus respectio, es necesario tener una referencia del cuadrado respectivo en una variable determinada, para asi luego poder determinar que color tiene el cuadrado, y asi llevar a cabo la tarea respectiva de asignar el bonus.AuraCristinaGarzon. En este link encontraran el historial de los respectivo a mi parte del proyecto Click

---

- Hola Aura

Tienes razon en lo del error, tienes que usar el codigo RGB(Red, Green, Blue)del color como esta en el image que hemos trabajado en clase, te recomendamos mirar el metodo WanderingLaberinto y ver las variaciones del color negro.

– JuanFelipeRiverosGuevara

- Hemos terminado nuestro metodo aqui esta el archivo .st donde se encuentra integrado el metodo de rebotar

mazesquadRebotar.st

- Hola, rebise el codigo para mazesquadRebotar y me alegra mucho que les haya servido el aporte que realize para esta parte del proyecto, felicitaciones porque hicieron un buen trabajo.AuraCristinaGarzon

---

- Hay un problema en el Ready.2 cuando uno dice crear laberinto en el boton crea es el laberinto de Ready.1 con el bonus en el laberinto y no deja pasar los robots.

Sergio_Navarrete

Es cierto, con respecto a eso, en esta pagina se encuentra el `.st` abria que agregarlo, sin embargo voy a añadirlo al ready.3.image que ya trae incluido el de construir puente.

- Debido a que existe comflictos en la integracion de el metodo del grupo de Laura,Jessica y Felipe, ya que el metodo no determina el color entonces siempre esta rebotando, entonces se tomo la alternativa que se cambia el tamaño del robot para que este se detubiera, y esto lo realizamos con el halo del robot y cambiar el tamño del mismo a el tamaño mas largo que el laberinto quedando de 75

CarlosRamirez

- Bien, quetal todos de nuevo, finalizando este curso ya he subido el `Ready.3.image` a la pagina del Internet Archive que esta en blanco pero con los botones correspondientes y contiene lo siguiente:

- Laberinto modificado (modificacion del bonus).

- Método de construir puente adaptado (boton constructor)

- Codigo de pasar de un corredor a la salida en el workspace

- El boton de parar, hace agrandar el robot (Falta que lo detenga)

–SergioAndresRivera

Hable con Offray y para solucionar el problema de hacerlo parar, me sugirio modificar el metodo de wanderingLaberinto para que en alguna parte preguntar si el tamaño del robot era distinto de 45 entonces que no avanzara. Como el boton `pared` lo que hace es agrandarlo, entonces usando el método `width` (o `height`), preguntamos si esto es mayor que 45, de esta forma si esto es cierto, hacemos que no avanze, y de lo contrario, que haga lo que normalmente es. Acá está la modificacion seleccionada(la que esta entre comillas):

```
wanderingLaberinto
        self comeToFront.
        self penColor: Color transparent.
        [self colorUnder = Color red]
                whileFalse: [self colorUnder
                                        = (Color
                                                        r: 0.501
                                                        g: 0.501
                                                        b: 0.501)
                                ifTrue: [’’’self width > 45
                                                ifTrue: [self go: 0]
                                                ifFalse: [self go: 10.’’’
                                                        self colorUnder
                                                                        = (Color
                                                                                        r: 0
                                                                                        g: 0
                                                                                        b: 0.004)
                                                                ifTrue: [self crearTabl1]]]
                                ifFalse: [self go: -10.
                                        self turnLeft: (-1 raisedToInteger: 2 atRandom)
                                                        * 90 atRandom].
                        (Delay forMilliseconds: 100) wait]
```

De esta forma ya el robot vagabundea, para y se agranda cuando se le da click al boton pared, construye puente con el boton constructor y cuando llega a la salida advierta que ha llegado luego de vagabundear un poco en la salida. Todo esto corresponderá al archivo `Ready.4.image` que ya esta en el Internet Archive y como veo las cosas será el image a presentar, aunque faltó lo del bonus que nunca se actualizo. —SergioAndresRivera

Primero que todo, quiero felicitar a todas las personas que han estado de lleno en la creación y desarrollo de los dos proyectos. Ahora bien, he estado haciendo unas pruebas con respecto al método "Wandering" dentro del laberinto para hacer la eliminación de el excesivo repetimiento de bucles de búsqueda. La cuestion es la siguiente (veo que alguien borro lo que se habia escrito antes aca), el robot empieza a rebotar muchas veces pues el angulo que tenemos definido para ello es demasiado pequeño, una solucion para que rrecorra el mayor espacio posible con menos cantidades de rebotes es disminuyendo ese àngulo. Ahora bien, uno de los problemas que tiene esto es la lentitud del robot para salir de los lugares donde el tunel es paralelo a otro lugar del mismo.

Hola, mi nombre es Juan Leguizamon, integrante del grupo Maze Squad 2. Les informo que gracias a la ayuda de Jose Ramires; se puede sacar un pequeño letrero, aunque aun no hemos podido encontrar un metodo que por cada robot tome un porcentaje y que cuando lleguen todos los robots aparesca un porcentaje final

Del metodo *wanderingLaberinto* se agrega un pequeño comando para que cada robot pueda sacar un porcentaje. Aunque el problema solamente radica en que no aparesca un porcentaje de cada robot cuando llega, sino que aparesca un porcentaje final (la sumatorias de cada porcentaje que tiene cada robot).

```
.....   self turnLeft: (-1 raisedToInteger: 0 atRandom)
                                                    * 90 atRandom].
                        (Delay forMilliseconds: 50) wait].
        self
                corredor: (Laberinto at: 15)
                salidaLaberinto: (Laberinto at: 16)
```

JuanLeguizamon

Me gustaria saber con presicion sobre cual Image trabajo, el mismo Offray dijo que se deberia trabajar sobre el ultimo que se ha publicado, y este fué el image.5, yo estoy trabajando sobre este y no me gustarian sorpresas a ultimo momento porfavor sobre que image trabajo?. Gracias por la atención a la presente. AuraCristinaGarzon.

El Ultimo que se subio fue el Ready.5.image, trabaja sobre esto. NO se si el grupo de sebastian leguizamon y sergio este trabajando sobre otro. Pero siempre usa el ultimo que se publico –- SergioAndresRivera

Muchas gracias Sergio, es que como en este historial si te das cuenta alguien no se quien, dice que el último image a trabajar es el .4, firma con tu nombre entonces depronto esto da pie a confusiones, pero muchas gracias por tu atención y así seguire trabajando sobre el .5 como tu lo dices. AuraCristinaGarzon

- Aura,

  Como acordamos en clase, los *images* son nombrados con números incrementales (3,4,5) a fin de dar cuenta de la última versión con la que se trabaja. Si mira los históricos y ha hecho seguimiento al foro, se dará cuenta que cuando Sergio publicó y firmo refiriéndose al image 4 es porque en ese instante no teníamos sino hasta la versión 3 y dado que se trataba de cambios mayores, debian ser integrados en un image con número posterior de acuerdo a la convención adoptada. En la medida en que vamos creando nuevas versiones debemos integrar el código a image con versiones mayores – offray 2007-05-23 20:42:52

Si Aura, debes guardarlo como un Ready.6.image en la misma pagina. Si tienes algun problema o algo asi, postea aca que hiciste al codigo y yo lo subo por terminal desde linux. SergioAndresRivera

- Otra posibilidad es traer el código st del método y hacer un *file in* del mismo dentro del image 5. Queda a su elección producir y subir un nuevo image, con ayuda de los compañeros o hacer el *file in* de una versión modificada del código st.Lo importante es que ese método funcione en el proyecto general. – offray 2007-05-23 20:42:52

Aura, por favor publica la categoria `mazeSquad.st` en tu pagina para integrarla al que va a ser el Ready.6.image. O sino sube ese `Ready.6.image` al Internet Archive para que podamos realizar todo. Se me hace que no es necesario poner el Transcript show y ademas las variables que declaras tienes que declararlas como variables globales (es decir que comienzen con Mayuscula). Puedes guardarlo todo en estas variables globales que serian tres, y luego en el método corredor: Salida: en vez de poner "llegué" en el PopUpMenu se podria poner "Mi puntaje es **P**untajetotal". OJO ACUERDATE DE DECLARAR VARIABLES GLOBALES Y PUBLICAR ALGUNA DE LAS DOS COSAS. –- SergioAndresRivera

---

Yo ya integre mis metodos de bonus con el Ready.5.image que esta disponible en el Internet archive. Supongo que en este ya todos tienen sus cambios respectivos. En lo que respecta al Transcript Show, es mas que necesario adecuado y pertienente usarlo, ya que a través de este se puede apreciar el cambio de valor de las variables respectivas, y asi comprobar con exactitud y precision el calculo de puntajes. OJO TU TAMBIEN, QUE NO HAS PUBLICADO ALGUN EJEMPLO DE COMO USAR VARIABLES GLOBALES EN BOTS INC. Ya que haces la observación respectiva, haz también el APORTE RESPECTIVO. (ya hice la prueba de usar variables globales, pero lo de las mayusculas no funciona). Y el libro no habla nada al respecto. Como Catolica y creyente, agradezco mucho a Dios por aportarme mucho con su bondad,gracia y sabiduria a la hora de hacer el trabajo respectivo del bonus, ya que nadie mas me aporto, como yo si lo hice.

Con respecto a la publicación de los métodos, ya están publicados. Si te queda algun tiempo, integralos con tu ready.5.image y me cuentas como te fue.

Aura procura no hablar tan golpeado con sarcasmos y comentarios de esa índole ya que las últimas clases no fuiste. No viene al caso ese "resentimiento" que tienes ya que todo es en pro de que el proyecto quede bien terminado no de quien hace mas o hace menos. En fin, si miras la documentacion de la pagina de nosotros (Click) en la semana correspondiente del 7 al 12 de Mayo esta documentado el aporte respectivo que llamas. Ademas eso lo hemos tratado en clase para adaptarlo a los botones y a la construccion del Laberinto. Respecto a lo del internet archive, el archivo es el Ready.5.image que fue el que yo subi (ya que la ultima modificacion que aparece es del 21 de Mayo) y no está modificado con lo que tu tienes, deberias subirlo como Ready.6.image. Si optas por otra opcion, entonces solamente es que publiques los métodos no que pongas capturas de pantalla, es decir, que cojas del microbrowser la categoria produccionMazesquad que es la que tiene todos los métodos, le des click derecho y exportes eso como un archivo.st. Luego lo subes, para que nosotros lo podamos adaptar.

Tambien he leido y al parecer en el PopUpMenu no se pueden poner los valores de alguna variable, entonces parece que tienes razon al usar el transcript.

Hola Soy Maria Fernanda Cubillos integrante del grupo encargado de realizar los puentes segun sugerencias de algunos compañeros mire el Ready.5 para verificar el funcionamiento normal del codigo de crear puente, porque el segundo bot caia en el hueco, pero

lo probe y el primero construye perfectamente y el otro no se cae de pronto el problema es que el segundo este muy cerca del otro mientras este construye el puente, pero de resto esta bien.

Realmente no recuerdo el orden pero haz el siguiente ejercicio haber si si funciona. Creas un laberinto, creas el robot 1 y luego el robot 2. El Bot 1 va a ser constructor, el otro será normal. Si todo sucede bien elimina todo el laberinto y vuelve a crear primero el robot 1 y luego el robot2, ahora el robot 2 va a ser el constructor y el Bot 1 sera el normal. Si todo funciona bien entonces es muy raro que a Aura y a Mi nos pase eso. –-SergioAndresRivera

Sergio hice lo que me dijiste y me di cuenta que es cuando el robot constructor llega que sucede que el otro se cae al hueco, pues no se por que pasara esto pero voy a mirar y mañana le comento a offray para ver que hacemos.

A bueno Maria Fernanda, igual en general el metodo funciona bien, solo que es un caso bastante extraño. Y adelante mañana lo comentamos fresca. –SergioAndresRivera

---

Finalmente he subido el `Ready.6.image` a este link. Incluye los métodos correspondientes a la parte de Aura aunque se le hicieron unas modificaciones bastante sutiles ya que cuando llegue a la salida arrojaba de una el Puntaje Total y no vagabundeaba en la salida como si lo ha hecho en los anteriores .images. Para ello dejamos el puntaje total para que lo muestro al final del Transcript, y el resto si lo dejamos igual. –-SergioAndresRivera

---

Estube mirando la ultima imagen que se subio la 6, encontre varios problemas que no se como se podrian resolver. Partiendo de que se crea primero el Bot1 y luego el 2, en este caso lo que veo es que en el momento de colocarle al bot 1 el ser pared lo hace perfectamente, pero si se hace en el bot 2 no lo hace, lo que hace es que el bot 1 le pega al bot 2 que esta siendo la pared y lo mueve. Por otro lado, estube observando que si el bot 2 construye el puente, lo pasa sin ningun problema pero que pasa, si el bot 1 va a pasar no lo deja sino lo que hace es caerse y pues la idea es que los dos pasen, si se intenta hacer de nuevo el puente con el otro bot si lo hace pero la idea no es repetirlo ademas crea el puente por debajo del que ya se creo. Estube mirando y pues la idea es que si lo hace en uno deberia hacerlo en el otro no se como arreglar ese problema seria bueno que miraramos y buscaramos una solucion para que todo sirva en cualquier robot. Gracias. –-JuanFelipeRiverosGuevara

Hola estuve mirando el Ready.6.image‘ e incontre ciertos problemas uuno de ellos es cuando los robot adquieren un bonus pero cuando se le da que el boton "Borrar" los robots se borran y uno puede volver a empezar pero ent en el Transcript queda el record del bonus anterior seria conveniente q cuando se le de borrar tambien borre el puntaje del bonus anterior tambien cuando el robot pasa por el bonus la sumatoria de cada robot deberia ser aparte para q sea mas legible por el usuario ya q la sumatoria es muy rapida y no se distingue el puntaje de cada uno y estoy deacuerdo con juan felipe pues yo tambien me di cuenta de ese error de construir el puente espero tengan encuanta las sugerencias como una critica constructiva para el preoyecto MazeSquad2.–- JessicaCastellanos

Mirando el archivo Ready.6.image ademas del error que encontro JessicaCastellanos; encontre lo siguiente: Cuando le digas a alguno de los dos robots, construir, y si alguno de los dos pasa por alguno de los bonus, el robot regresa al estado normal. No se si es un error o si en verdad asi debe ser el juego.

JuanLeguizamon

Al parecer creo que estas fallas tienen que ver a que un robot se construye primero que el otro, es decir, uno de los dos queda por encima, alterando todos los métodos para ese robot, en el otro funcionan bien. Respecto a lo que dice Jessica en lo del bonus creo que si deberia pasar eso que ella menciona de que cuando se borran los bots tambien se borren los puntajes. Si nada mas se usa un transcript cuando ambos robots llegan a la salida deberian mostrar cada uno en el transcript el puntaje que obtuvo, pero como vemos, lo que se obtiene es repetir el puntaje. Es decir, lo que pasa al parecer es que el bonus esta establecido como un puntaje general, no como un puntaje especifico para cada robot, como deberia ser ya que dentro del juego asic omo cada robot puede construir un puente, vagar, parar, volver a su estado normal, tambien deberia llevar su propio puntaje. –- SergioAndresRivera

---

Tengo entendido que JoseRamirez creo el boton de borrar quisiera saber q metodo utilizo o donde se puede ver la categoria,scrip o lo q haya utilizado para el funcionamiento del boton `borrar` para poder lograr obtener la solucion del problema que plantee anteriormente. gracias –- JessicaCastellanos

JoseRamirez: Hola, lo que pasa es que el boton borrar practicamente no tiene codigo, tome un workspace, para luego seleccionarle el boton de clear all, para continuar con los pasos que publique en el historia de produccion robots. por lo tanto este boton no contiene un codigo en especifico.

Hola a todos:

En cuanto a lo del Ready.6 la encargada de hacer esta tarea fuí yo AuraCristinaGarzon, y efectivamente así lo hice, por otra parte en cuanto al transcrip utilizado para los puntajes de bonus, es una trabajo extra que yo realize y tal como lo dicen funciona y arroja un puntaje para los dos robots conjuntamente, me parece que así como hay errores con el metodo de construir puente y otros que no funcionan para cada robot por separado, es pertinente que el que quiera aportar para pulir esto lo puede hacer; tambien es importante destacar el hecho que yo fuí la persona que subio el archivo ready.6.image a la pagina con todos los metodos integrados, y en este se encuentra ya la modificaciòn o integración del bonus azulclaro(+20) y amarillo(-10), el cual funciona como un antibonus, es decir que el robot al pasar por este cuadro no gana puntos si no disminuye puntos; esto se hizo con la utilizacion como ya lo habia mencionado del respectivo transcrip; espero que esta idea sea de mucho valor para la persona que desee pulir el proyecto. En el siguiente Link podran apreciar todo el trabajo que me lllevo para llegar a la integracion de esta parte de bonus al proyecto de MazeSquad2, ahi podrán observar todos mis intentos en lograr que el robot reconociera el color objetivo que se había puesto en el Laberinto Standar(Rosado) y se podran dar cuenta que finalmente este color el programa Squeak nunca lo reconocio, es por esto que decidí colocarlo de otro color pero ademas añadí un nuevo bonus en otra ubicaión, obteniendo como resultado final lo que pudieron observar en el ready.6.image final, y que obviamente pueden pulir, por ejemplo en lo referente al transcrip para cada robot por separado, el puente que verdaderamente sea un puente que sirva para los dos robots, la pared que funciones como debe ser, etc; afortunadamente son cosas se pueden solucionar con mucho trabajo, esfuerzo y dedicacion, así como yo logre todo lo que realize en este proyecto. Muchas gracias por sus comentarios. En el siguiente link encontraran todo el historial que fué publicado el dia 25 de mayo casi todo el día Click. AuraCristinaGarzon

MazeSquad/Foro (last edited 2007-05-27 19:04:17 by offray)