Road To Civilization

Proyecto asignaturas de **Simulación**, **Inteligencia Artificial** y **Compilación** del 3er año de Ciencias de la Computación de la Universidad de la Habana.

Desarrolladores:

- Alejandro Escobar Giraudy (C312)
- Airelys Collazo Pérez (C312)
- Henri Daniel Peña Dequero (C311)

1. Introducción al problema y búsqueda de una posible solución

Como bien el título ya da a entender, nuestra propuesta es un Simulador de Civilizaciones, a partir del uso de herramientas de las tres asignaturas. Nuestra idea inicial es hacerlo por capas de trabajo, donde desarrollaremos una serie de características y funcionalidades al programa una vez la base del proyecto esté completa.

La idea base del proyecto es la siguiente:

1. Crear un simulador que nos permita analizar el comportamiento de una civilización, dadas las condiciones de un terreno, y su evolución con el paso del tiempo.
2. La civilización estará dotada de una serie de variables que explican su funcionamiento y avance, dígase el índice de mortalidad, natalidad, gestación, promedio de vida, avance tecnológico, potencial adaptativo (dependiendo del medio ambiente definido), etc, y estará dotada de un comportamiento, dígase tendencia de movimiento, agresividad, actividades laborales principales, etc.
3. Igualmente será posible establecer especies de menor o mayor alcance con respecto a una civilización, o sea, será posible establecer a especies atrasadas (ejemplo: moluscos, artrópodos, etc), especies intermedias (ejemplo: perros, gatos) yuna especie avanzada(ejemplo: humanos).

La idea no es que se establezcan en el nivel según los ejemplos anteriores, sino que, a partir de un lenguaje definido por nosotros y un compilador para el mismo, puedan ser creadas especies con gran cantidad de características, incluso que el usuario defina el comportamiento de la mayor parte de las variables.

Propondremos realizar la evolución de la civilización en el tiempo utilizando herramientas de inteligencia artificial, así como el uso de estas herramientas en otras partes del trabajo a medida que avancemos.Una vez que cumplamos con la base de nuestro simulador, nos tomaremos como meta lograr las siguientes ideas por capas:

- Implementar un sistema de desastres naturales, dígase huracanes, terremotos, etc.
- Implementar un sistema de enfermedades naturales, ya hereditarias, infecciosas.
- Implementar un sistema de ambientación. Esto incluirá terrenos con características climáticas, estaciones, etc.-Implementar la posibilidad de permitir diferentes intervalos de tiempos de avance en la simulación (ejemplo: días, semanas, meses, años, décadas, etc).
- Lograr la existencia de más de una civilización, lo cual da paso a establecer un sistema de interacción entre dos civilizaciones distintas (dígase problemas menores, guerras, alianzas, uniones, etc)
- Posibilitar la transición entre el estado de evolución de una especie, dígase de atrasado a intermedio, o de intermedio a civilizado.
- Habilitar la existencia de subgrupos dentro de una misma especie. Sectores dedicados a cierta parte de la civilización (ejemplo: obreros, filósofos, artesanos, etc), grupos problemáticos, grupos desertores, etc.
- Habilitar la evolución de las enfermedades, adaptación, surgimiento, cambio (tanto de enfermedades hereditarias como infecciosas).
- Habilitar el cambio climático y evolución de este.Si bien el anterior orden de las capas de trabajo no es definitivo, en un primer momento esperaríamos fluyera de esa forma.

Queremos ser capaces de lograr que todas las capas de trabajo mencionadas puedan ser generadas a partir del lenguaje por el usuario (crear especies, subgrupos, desastres, enfermedades, etc) con las variables y evolución de las mismas que se estime conveniente.

El lenguaje, aún no definido claro está, lograremos que no se limite a la creación de la simulación, sinoquetambién pueda ser capaz de modificar dinámicamente el comportamiento y evolución de todo lo que es posible implementar en esta simulación.

Concretamente queremos crear un software que a nivel de problaciones de especies en un ambiente definido pueda simular el desarrollo de

las mismas de tal manera que podamos obtener,a través de técnicas de visualización, las estadísticas de losparámetros de interés en eltiempo y espacio que querramos conociendo todas las interacciones entre la disímiles especiesy/o poblacionesy el medio en que encuentran, además un entorno controlable en cualquier momento a partir del lenguaje creado específicamente para nuestro problema.

Espereamos la idea sea de su agrado, muchas gracias.

2. Primeros pasos

Se ha implementado un prototipo inicial del proyecto en el que esta definida tres clases principales: Specie, Land y Simulation. Estas componen las escencia de lo que se espera como producto final.
Specie define cada especie por separado, esta consta con un nombre, una lista de características(las que tienen por valor un número o un rango), y otra de dependencias entre las mismas donde podemos definir las relaciones entre las características de la siguiente forma:
a -> b * c
Al pasar un día en la simulación esta dependencia es expresada de la siguienteforma:
b += a * c

Land define una parte del terreno, esta consta de una lista de características, y otra de dependencias siguiendo la misma lógica vista anteriormente

Simulation se encarga de contener el terreno, crear las especies, además esta debe contener las reglas de nuestro sistema y de acuerdo a a las mismas correr las simulación.

2.1 Problemáticas principales

Definir las reglas sobre como va a estar distribuido las especies en el terreno y las consecuencias de tal forma que si crece la población como se ditribuye en el terreno, o si lo hacemos región a región

Otra problemática principal es como definir las interrelaciones terreno - terreno, especie - especie, terreno - specie, specie - terreno. Puesto que solo definimos las dependencias entre la misma especie o terreno, pero como definimos entre instancias diferentes o entre terrenos y especies. este problema necesita más debate entre los desarrolladores y se espera llegar a un consenso en el próximo paso

Definir las reglas de nuestra simulación que no son modificables por el usuario, vitales para un correco funcionamiento del sistema.

2.2 Proximos pasos

Lograr un consenso para definir la resolución de las problemáticas y llevarlo al proyecto.

Tener una primera gramática y lograr un avance significativo en la defición del lenguaje a usar por el usuario, así como el tokenizador, parser, etc

Implementar las funciones de distribución a usar dado que actualmente solo se hace un random por defecto y por supuesto eso se debe mejorar.

Perfeccionar la simulación dado que esta corre aplicando als dependencias una a una, pero no como un conjunto que sucede todo al mismo tiempo, creemos que ir introduciendo ecuaciones diferenciales nos puede ayudar a lograrlo

Crear estructuras y algoritmos para una optimización de lo ya implementado

3. Restructurando mundo, implementando reglas y adaptando estructura por capas.

En esta iteracion se ha avanzado en resolver las problematicas encontradas en el anterior modelado del problema conllevnado consigui una restrucutracion parcial del proceto, un cambio de filosofia en la implememtacion.

3.1. Primeros cambios

Nuevo:

- Concepto de sociedad como unidad atomica, agente en esta simulacion que busca supervivencia, esta pertenece a una unica especie y se ubica en un unico terreno.

- Concepto de influencia, dado que las dependencias son hechos que ocurren en un inervalo de tiempo, es necesario tener un hecho que represente una correspondencia entre dos caracteristicas de tal manera que cuando una cambie la otra cambie proporcional al cambio de la primera. Ejemplo: A mayor economia, menor hambre. Como dependencia este no se podria representar dado que solo logramos que en cada iteracion aumente proporcional a toda la economia, ahora con influencias logramos una correcta modelacion de la problematica dado que hambre aumenta o disminuye propoircional al cambio que tenga la economia esa iteracion.
- Limites en las caracteristicas, importante para lograr una facil y correcta modelacion de algunas caracteristicas por ejemplo la cantidad de los recursos o el agua dado que no puede ser menor que 0.
- Lista de entidades por terreno, en la que guardamos todas las entidades que interactuan en el mismo.
- Lista de operaciones y distribuciones en la que guardamos las operaciones que se hacen entre caracteristicas(dependencias, influencias), y las distribuciones que se calculan al trabajar en los rangos.

Cambios:

- Specie reconvertido a un simple contenedor por ahora, el objetivo es que persista como concepto global, que se pueda apoyar en ello para obtener las salidas a estudiar pero que como tal no tenga como generar ninguna accion en la simulacion.
- Redefinicion de los conceptos de dependencias e interdependencias, dependencias son todas las dependencias e influecias que ocurren a nivel de terreno, ya sea en el mismos terrenos, en una sociedad o entre sociedades o sociedad - terreno, interdependenias son todo cambio que ocurre a nivel de simulacion, de terreno a terreno(entre los mismos terrenos o sus sociedades)
- Calculo de dependencias usando la lista de operaciones y distribuciones.
- Resuelto problemas de importancia de orden de las dependecias al calcular el move_one_day.

3.2 Mejorias de los cambios

Tras esta nueva iteracion se vienen varias mejoras, empezando con la inclusión del concepto sociedad y reestructuracion del proyecto dado que primeramente resuleve el problema de que las especies actuaban en general, no habia diferencias de cuantos pudieran estar en un terreno y las consecuencias que esto implica con respecto a las inter-dependencias, esto representaba una inconsistencia que con la inclusión de las sociedades queda resuelto, ademas tenemos una estructura mas sólida, y las socedades se convierten asi en la unidad atomica del proyecto sobre las que queremos construir el concepto de agentes en busca de superviviencia.

Además se agrega dos conceptos de suma importancia para darle la libertad al usuario de escribir un modelo mas acorde a lo que se espera teniendo dos reglas a su disposicion como son las influencias y los limites explicados anteriormente.

También se resuelve un problema que es la importancia del orden de las dependencias a la hora de ejecutar el metodo move_one_day, esto hace que sea una simulacion mas justa dado que en este modelo todo no puede pasar al mismo tiempo al menos nos olvidamos de que orden se van calculando las dependencias y que este influyene en la simulacion o no, solo bastaba con gurdar los cambios en una lista aparte y luego actualizarlo.

Otro importante cambio fue en la construccion de lista de operaciones para calcular las dependencias e influencias lo que nos permite una mayor abstraccion dando paso que el usuario pueda definir su propia operacion o distribucion y añadirla a la la lista para luego usarla.

Especies ahora es un objeto abstracto, que es completamente dependiente de las sociedades de la misma que exista en ese momento. Esto nos ofrece la posibilidad de poder actuar y revisar las especies de forma mas general independiente del lugar en que pueda encontrar sociedades.

3.3 Proximos pasos

- Terminar el compilador
- Implementar Mutacion y Migracion

4. Algoritmo de Inteligencia Artificial: Evolución de las sociedades

En esta nueva actualización logramos completar el primer algoritmo de Inteligencia Artificial en el Proyecto, enfocado en lograr el concepto de "Evolución" de las especies.

La idea es simular un desconocimiento por parte de las sociedades, quienes solo tendrán acceso a sus datos (características y dependencias) y comenzarán a intentar descubrir cómo funcionan las dependencias que influyen en ellos y que sin embargo pertenecen a factores externos (ya sea de un Land u otras sociedades).

Society comenzará a guardar información de aquellas características ajenas que afecten a sus propias características, esta información que

guarda será limitada, por lo que tendrá que comenzar a deducir el funcionamiento. Supongamos que tenemos una sociedad "Humanos", que se encuentran en un Land que afecta a su característica "Población" de manera positiva con "Oxígeno" y de manera negativa con "Frío", nuestra sociedad sabrá que le afectan dos características, sabe a quién pertenece (en este caso es el Land en el que se encuentra), pero no sabe cuánto le afecta cada una. Solo tiene conocimiento de una cosa, en general cuánto ha bajado o subido su característica "Población" con respecto al día anterior. Si tomáramos como ejemplo más exacto que "Frío" disminuye la Población en 10 y que Oxígeno la aumenta en 3, "Humanos" saben que hubo un aumento de 7 en "Población" con respecto al adía anterior, pero como ya dijimos, no está seguro de cuál de las dos aumenta y cuál de las dos disminuye.

La idea será que comenzará a pedir Request a Simulation sobre crear nuevos nodos que afecten a estas características. ¿Cómo sabe si las tiene que afectar positiva o negativamente?, pues tendrá que descubrirlo a prueba y error haciendo Request distintos y observando si las características que quería mejorar realmente mejoran, o si cometieron un error.

Claro, las sociedades no son dioses, no pueden pedir Request y esperar que serán fácilmente atendidos, tendrán que luchar por ello.

Aquí entra el juego el concepto de "Mutación", que es una nueva variable que tendrán las Características a partir de ahora, y representa la probabilidad de que un Request enviado por una sociedad para afectar una característica sea aceptado. Por ejemplo, el nivel de Mutación de "Frío" de un Land con ambiente poco hostial, pudiera ser bajo en comparación con el de "Tamaño" de una montaña, puesto que es más sencillo abrigarse del frío que modificar la forma de la montaña en sí.

Ahora, las sociedades no conocen el valor de la mutación de una característica externa a las suyas, por tanto, deberá realizar distintos intentos para modificar una característica antes de darse cuenta de que quizás es muy poco probable de que lo logren (ya que son inteligentes, y no obstinados, y no van a intentar infinitamente modificar una característica que quizás, pudiera tener incluso 0% de probabilidades de ser modificada).

En caso de lograrlo, la sociedad analizará si la decisión tomada fue buena y ha mejorado las características suyas, o si fue una desición errónea, en este caso tiene la capacidad de cambiar su Request para probar afectar a la característica de forma inversa, y nuevamente analizar si su análisis fue correcto, y en caso de que por ninguna de las dos vías pueda llegar a una conclusión válida, retirar el Request y concentrarse en modificar otras características.

Falta solo una cosa por decidir, ¿qué característica de las que me conforman mejoro primero?, pues este algoritmo está diseñado para priorizar la única característica obligatoria y además, indispoensable para la supervivencia de la sociedad, la "Población", y a partir de ahí, viajando por sus propias características e interdependencias puede ir señalando cuáles de sus propias características son negativas y cuáles positivas con respectoa a su Población.

Desde el punto de vista matemático esto puede ser concebido como un problema de optimización, teniendo en cuenta mejorar aquellas características que sean positivas con respecto a Población y disminuir aquellas que le afecten de manera negativa.

5. Características de la Simulación.

Hay que analizar ahora con qué cosas cumple nuestra Simulación sobre la base de los contenidos obtenidos en la asignatura. Las características principales que podemos observar son:

- Con la adición del sistema de Evolución de las Sociedades, estas comienzan a interactuar con el medio ambiente (el simulador y los land), y son capaces de comenzar a tomar decisiones con respecto a qué nodos afectar en la simulación, ya no quedan solo como contenedores inmóviles esperando a que otra característica les afecte, pudiendo entonces señalarse formalmente que las sociedades pasan a ser los agentes de la simulación, y además, agentes inteligentes y autónomos, puesto que toman sus decisiones buscando la mejor solución posible, y además no dependen del ser humano para tomar estas deciciones una vez comenzada la Simulación.
- Los agentes no tienen dominio sobre el medio ambiente, e incluso pueden fallar constantemente en sus decisiones, por tanto, estamos en presencia de un ambiente no determinista. Podemos verlo también como que, dada una acción del agente, no tenemos total seguridad de qué puede ocurrir en la simulación.
- El ambiente es innacesible, puesto que el agente no obtiene libremente información sobre los datos del mismo, solo de una parte, que son las características que les afectan, y no de todos los datos de estas características.
- El ambiente es dinámico, puesto que, ajeno a las acciones del agente, pueden ocurrir cambios en la simulación, ya sea que por características naturales (dependencias del land a las sociedades) sean elimindas algunas sociedades, o en menor medida sean afectadas sus características, ajeno a sus Request
- El ambiente es discreto, puesto que es conocido el número de dependencias e influencias que serán resueltas en un día que pase en la simulación, y se sbae que cada sociedad pedirá a lo sumo un Request
- Los agentes son reactivos, ellos aprenden del ambiente y actualizan los datos ante cualquier cambio que ocurra con respecto a ellos, son proactivos, puesto que intentan optimizar en todo momento sus características independientemente del estado del ambiente, y son

sociables, puesto que pueden interactuar con otros agentes a través de las dependencias, ya sea ayudándolos a partir de un beneficio mutuo, o atacándolos en vista de una amenaza, formalizando el hecho de que son agentes propiamente inteligentes.

5.1

Adicionalmente Species fue modificado como una clase general que engloba todos los valores de las características de las distintas sociedades a la que pertenecen. Así es posible obtener la información general de todas las sociedades que pertenezcan a una misma especie, por ejemplo, la sumatoria de todas las poblaciones de todas las sociedades de "Humano" se guardarían en la especie de igual nombre, y así con todas las características que compartan.

Por otro lado, Características y Dependencias fueron modificados para pasar a ser clases propiamente en sus .py específicos, así permiten el uso de referencias y un fácil acceso a sus datos.

6 Compilador

Para hacer el compilador inicialmente se procede a hacer el lenguaje el cuál se tiene definido qué estructura básica tiene.
El mismo definido ya en el documento "propuesta de lenguaje" posee todos los recursos para crear e interactuar con la simulación además de permitir libertad al usuario de hacer funciones, usar variables y estructuras de control como comúnmente se usa en otros lenguajes de programación.

Antes de continuar, primero es conveniente hacer el lexer del que depende los otros pasos, se hizo un lexer sin uso de librerías, para ello se construyen las expresiones regulares con un intérprete de expresiones regulares hecho por los desarrolladores. Adicionalmente se hizo otro lexer haciendo uso de la librería re, por una cuestión de legibilidad y rapidez dado que esta librería ya fue optimizada. Ambos lexers tienen una tabla con las expresiones regulares que definen cada parte de nuestro lenguaje, y se matchea la entrada devolviendo tuplas con el lexema, tipo y línea, ejemplo: (var, variable, 1), ("hola", string, 23), (_factorial, function_name, 3).

Luego de tener una idea de que lenguaje se quiere se procede a crear una gramática, la que costó mucho trabajo y fue modificada varias veces en el desarrollo del proyecto. Para la misma se usó la clase Grammar, con la que primeramente se define el símbolo distinguido para indicicar el inicio; luego los "termianles", aquellos que no producen otras producciones, ejemplo de estos son: true, string, var, if...; seguido se definen los "no terminales" que producen partes de las cadenas permitidas en el lenguaje, por ejmeplo: arithmetic_expresion, argument_list...; por último se declara todas las producciones por cada "no terminal". Es de tener en cuenta que esta debe cumplir con que se elimine recursión izquierda, asocie adecuadamente y demás características vistas en conferencias.

Una vez finalizada la gramática se procede a hacer el parser, el que nos crea las tablas *goto* y *action*, el mismo algortimo define si la gramática construida cumple con ese parser y en donde no lo cumple, que a partir de los token que genera el lexer, se hace uso de las anteriormente mencionadas para verificar que la entrada cumple con la sintaxis del lenguaje definido por la gramática, el parser hecho es el LR(1)

El ast es una jerarquía de clases construida a partir de la gramática, tiene nodos por acciones que permite el lenguaje. Este es usado luego para realizar el checkeo semántico y para la transpilación pues el ast guarda la escencia o estructura de lo que se va a ejecutar.

El checkeo semántico es el próximo paso a ejecutar, este se hace con el objetico de verificar que tenga sentido el programa a ejecutar, haciendo varios recorriedos sobre el ast verificando los tipos, variables existentes en cada nivel del código, o incluso para la inferencia de tipos aunque no es necesario en este caso. El primer recorrido hecho es el type_collector, dónde se guarda en el contexto los tipos definidos de nuestro lenguaje y las funciones predefinidas, el usuario no crea tipos nuevos por lo que no es necesario guardarlos. El segundo es el type_builder, que se encarga de recoger las definiciones de variables, funciones y argumentos de las mismas. Por último se hace el type_checker que se encarga de verificar que los tipos coincidan en asignaciones y declaraciones, también verifica que las variables asignadas se encuentren en el scope locar o en los superiores asegurando que una variable declarada en una función no se use afuera de la misma, que los retornos de funciones coincidan con el tipo de la misma, que los argumentos coincidan con la cantidad de que se predefinió en el type_builder o en el type_checker, entre otros errores semánticos, capturando el primero encontrado y saliendo así del recorrido en caso de que exista alguno.

Luego de verificar todo lo anterior mencionado se procede a transpilar el programa, para ello se hace otro recorrido en el ast, guardando por cada nodo un string del código que este ejecuta en python.

Al finalizar el recorrido se tiene un string con código en python listo para ser ejecutado con la función build in de python **exec**.

Luego este se ejecuta en el runner, archivo que reune las condiciones para que el programa tenga sentido, conteniendo las importaciones necesarias, así como los métodos del lenguajes ya implementados.

7 Conclusiones

Si bien el proyecto fue complejo en su mayoría, logramos cumplir con la mayoría de nuestros objetivos, esperamos que este proyecto pueda ser de ayuda y más adelante le sean implementados aquellas funcionalidades que pueden ayudar a darle mayor utilidad para ramas como la Biología por ejemplo. Esperamos haya sido de su agrado.