Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Modelación de Sistemas Multiagentes con Gráficas Computacionales (Gpo 102)

RETO AD2023_ROBOTS LIMPIADORES

Profesores:

Pedro Oscar Pérez Murueta
Alejandro Fernández
Denisse Lizbeth Maldonado Flores

Integrantes del equipo:

Ian Joab Padrón Corona | A01708940
Uri Jared Gopar Morales | A01709413
María Fernanda Moreno Gómez | A01708653

Miércoles 06 de septiembre de 2023

RETO AD2023_ROBOTS LIMPIADORES

Introducción:

La gestión de los desechos es un problema que atañe a todos a nivel mundial, es un problema que deriva desde el daño al medio ambiente, al cambio climático, causando dificultad para el desarrollo económico de países pobres y ricos. De acuerdo a un informe llamado *What a Waste 2.0,* en el mundo se generan anualmente 2010 millones de toneladas de desechos sólidos municipales y al menos el 33% de ellos no se gestionan correctamente (Banco Mundial, 2018). Se provee un aumento en la cantidad de basura en los próximos 25 años, por lo cual, es de bastante importancia el actuar para tener una buena gestión de desechos y así, prolongar nuestra propia existencia en este mundo.

Por ello, se planteó como reto local el gestionar los desechos de una oficina, de manera que se cuenta con 5 robots cuya tarea principal es recoger la basura en una oficina de ciertas dimensiones y llevarla al bote de basura, con la condición de que cada uno de ellos puede cargar a lo máximo 5 basuras por robot. Los robots no saben dónde están las basuras, esto solo lo puede hacer explorando las celdas que están a su alrededor, tomando en cuenta que existen obstáculos y estos robots tienen que rodearlos, ya que no pueden pasar encima de estos.

La oficina cuenta con una papelera con capacidad infinita y todos los robots conocen en qué lugar se encuentra, y por cada vez que los robots llenen su capacidad de almacenamiento, estos se van a dirigir a la papelera para vaciarlo, para que de esta manera puedan seguir limpiado y acabar con su trabajo. Nuestro reto es, crear un programa que haga la exploración del entorno y la recogida de basura en los menos pasos posibles, de manera que se ahorra energía y tiempo.

Herramientas implementadas:

- Lenguajes: Python, C#
    - Utilizamos Python para modelar la parte de multiagentes debido a su flexibilidad, facilidad de uso y una gran variedad de bibliotecas disponibles. Es fácil implementar agentes autónomos usando clases y objetos, por lo que

es fácil modelar agentes, además de que es fácil visualizar multiagentes con la biblioteca de matplotlib, tiene buenas bibliotecas para el manejo de datos como numpy y pandas, por lo que es muy versátil para trabajar.

    - Utilizamos C# para modelar la parte de gráficas computacionales en Unity, esto debido a su facilidad en sintaxis en la creación de videojuegos, tiene mucha documentación, es multiplataforma y tiene muchas librerías disponibles.

- Simulación: Unity
    - Seleccionamos Unity en primera, porque fue la plataforma de desarrollo de juegos y simulaciones en 2D y 3D que vimos en clase, además de que tiene un gran motor gráfico 3D y buenas herramientas de edición.
- Librerías: Mesa, Matplotlib, Numpy, Random.
    - Utilizamos Mesa para crear y simular agentes. Es muy útil, ya que facilitó la interacción entre agentes y la búsqueda en su entorno.
    - Usamos Matplotlib para crear los gráficos en la parte de Python, pudiendo crear nuestro gráfico en 2D primeramente aquí para poder evaluar el comportamiento de los agentes.
    - Numpy lo usamos para manejar valores numéricos, porque estamos constantemente usando operaciones numéricas para las basuras.
    - Random es útil para escoger una casilla aleatoria de las posiciones a las que el robot se puede mover.
- Espacio de desarrollo: Visual Studio Code

Solución:

Para lograr la solución al presente problema, comenzamos por decidir que cada uno de los objetos representados en el mapa serían agentes de la librería Mesa (Litter, Wall, PaperBin, Robot). Esto con la finalidad de que tuvieran un atributo con su tipo de agente, y así mismo poder diferenciarlos de una forma más rápida. Además, nos facilita la exploración que programamos para los robots, para que supieran que es lo que se encuentra en las celdas visitadas o a su alrededor (para el caso de muros). Lógicamente, el agente con mayor cantidad de atributos son los robots, ya que tienen las tareas de exploración, recolección

evitar los obstáculos, etc. Dadas estas consideraciones, dividiremos nuestra lógica en varias etapas.

Etapa Uno: Exploración:

Para esta primera etapa decidimos que los robots comienzan con movimientos aleatorios por todo el mapa. Tomando en cuenta la descripción del problema, los robots no pueden estar en la misma celda ocupada por otro robot después de salir de su celda de origen y si se topa con un obstáculo deberán rodearlos. Los movimientos aleatorios que hacen los robots consideran esto. Al mismo tiempo, estos movimientos continúan por n steps, donde n es igual al número de celdas que existe en el mapa. A través de varias iteraciones descubrimos que, al terminar esta etapa, los robots exploraban entre el 60 y 70% del mapa.

Los robots cuentan con un mapa compartido en el modelo (el cual se inicializa en blanco) con la finalidad que, con cada nueva celda explorada por los robots, se revelara su contenido y se actualizara el mapa. Al inicializar los objetos representados en el mapa como agentes y ponerles de atributo el tipo el robot es capaz de saber cuál agente encontró (basura, papelera, pared, u otro robot).

Al finalizar los movimientos aleatorios, con ayuda de su mapa interno pueden saber las celdas que todavía faltan por explorar. Estas celdas serán alcanzadas por los robots con la ayuda de un algoritmo de búsqueda BFS (Breadth First Search), lo que hace es que de las posiciones donde se encuentran los robots buscan el camino más corto para estas celdas no exploradas. De esta manera son capaces de poder explorar todo el mapa y saber que existe en cada celda de este.

Etapa Dos: Recolección

Acabada la etapa de exploración, entrará la etapa de recolección. Al tener el mapa explorado y los tipos de agentes que existen en cada celda, es momento de que los robots se muevan a cada celda que concuerde con el tipo de agente de la basura. Para ello, el programa primero itera sobre toda la matriz de celdas exploradas y obtiene las posiciones donde los robots encontraron basura, para después agregarlas a una lista.

Una vez reconocidas las celdas que cuentan con basura, los robots invocarán una función para determinar cuál de estas celdas es la más cercana a su posición actual. Con esto podrán

ir hacia esa celda y comenzarán a recolectar la basura que puedan (En base a la capacidad que tienen en ese momento). Para calcular estos movimientos los robots vuelven a hacer una búsqueda con nuestro algoritmo de BFS, trazando el camino óptimo hacia dichas celdas.

Posteriormente, los robots “eliminarán” los agentes del tipo basura de estas celdas y aumentarán su contador interno de basura (lo cual referimos como el proceso de recolección). Si los robots terminan de recolectar la basura de una celda antes de ir a la papelera, y todavía les queda espacio, irán a otra celda cercana por más basura. Una vez cumplan con su capacidad de 5 basuras, los robots van a la papelera para dejar toda la basura recolectada (en términos del código, es aquí cuando los robots resetean su contador de basura a 0).

Al llegar a la papelera, los robots verifican si todavía existe basura en la celda de la que provenían, lo cual significa que no lograron limpiar toda la basura de su celda. Si esto es cierto, estos volverán a esta celda para acabar con la basura. Esta funcionalidad es posible ya que cada robot tendrá una memoria de la celda en la que se encontraba y si se vació cuando recogió basura de esa celda, por lo que si llega a 0 esta memoria se borra. En caso de que no sea cierto, significa que ya terminó de recoger basura en la celda que tenía asignada ese robot, por lo que se le asignará una nueva celda de la lista que se planteó al inicio de esta etapa.

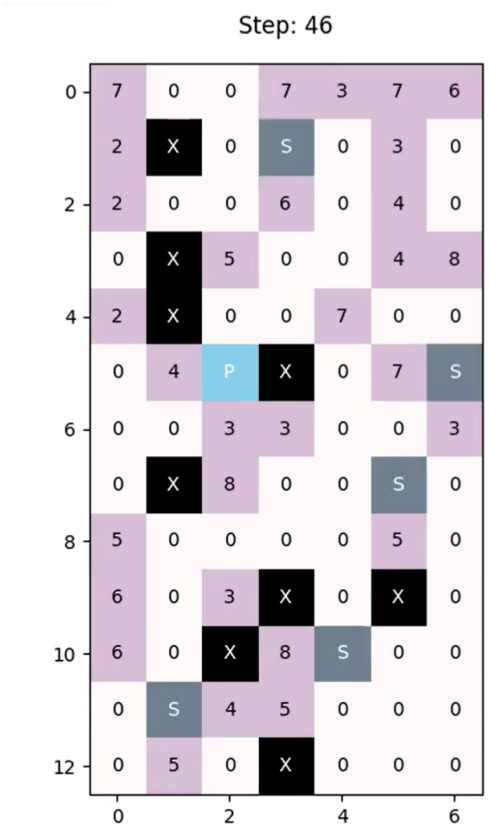

Imagen 1.0. Representación de los robots en una gráfica de Python, utilizando matplotlib para graficar. Las “X” son los obstáculos, los números, la cantidad de basura, “P” la papelera y “S” son los robots.

Etapa Tres: Finalización

Finalmente, cuando un robot ya no puede ir a una celda con basura, (debido a que ya hay otros robots asignados a estas o ya no hay más en la lista de celdas con basura), estos comenzarán a moverse aleatoriamente por el mapa (sin pasar por la papelera). en espera de que sus compañeros robots terminen de dejar la basura restante a la papelera. Al finalizar todos los robots, el programa terminará, dejando saber al usuario la cantidad de steps que les tomó a los robots terminar su misión de limpieza.

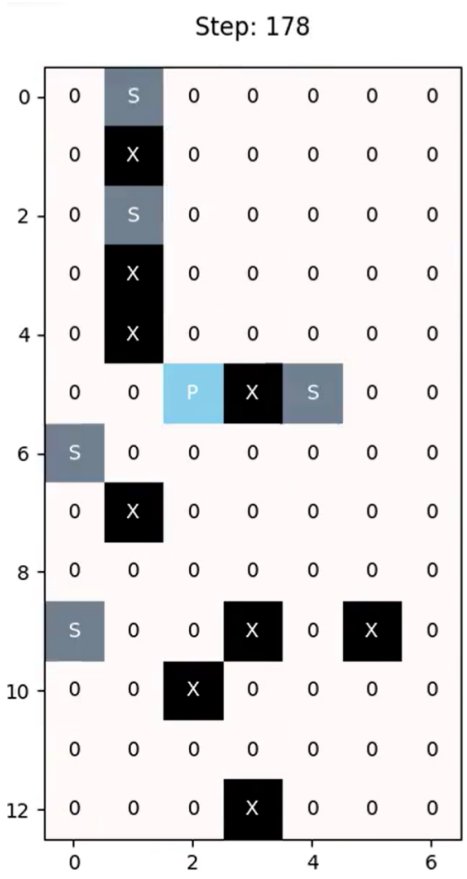

Imagen 1.1. Al término de que los robots dejan de recolectar la basura, el grid toma el valor de 0 en aquellas celdas donde había basura, los últimos robots dejan las últimas basuras y si hay robots que ya terminaron y no tienen basuras por recoger, hacen movimientos aleatorios hasta que los robots con basura la dejen en la papelera.

Unity

Primeramente, hacemos un proyecto en 3D en Unity y buscamos los assets deseados para representar la basura, los obstáculos, los robots y el piso.

Descargamos los siguientes de la Unity Store:

1. LowPoly Server Room Props by iPoly3D (Para utilizar el prefab del servidor)
2. Cola Can by Rodolfo Rubens (Para utilizar el prefab de la basura)
3. Jammo Character | Mix and Jam by Mix and Jam (Para utilizar el prefab del robot)
4. Trash Low Poly Cartoon Pack by BlankfaceStanislav (Para utilizar el prefab del bote de basura)

5. 25+ Free Stylized Textures by Game Buffs (Para utilizar la textura del piso de mármol)

Una vez con estos descargados, los importamos a nuestros assets, y posteriormente, definimos los valores predeterminados para poder convertirlos en prefabs. Para hacer los prefabs de basura, se generarán dos montañas de 4 latas, cada una para el valor más alto de basura que es 8, y para los otros, se les va restando una lata hasta llegar a 1, y se hacen prefabs para poder instanciarlos varias veces sin tener que volver a asignar sus valores.

Para hacer el piso (Tile) simplemente se agrega un cubo a la escena, y se le da la forma aplanada de piso. Para darle un toque más lindo, se le agregó la textura de mármol que anteriormente descargamos, definimos las dimensiones deseadas del piso y lo hacemos prefab.

Imagen 1.2. Visualización de la carpeta Prefabs en Assets

Una vez con los prefabs, se puede empezar a codificar. Se hizo una carpeta de Scripts en Assets (para almacenar todos los scripts que utilizará nuestra simulación) y agregamos los siguientes:

- **CameraController.cs:** Este sirve para poder mover la cámara en el modo Game hacia arriba, hacia abajo, derecha e izquierda usando A,W,S,D, así como el rotar la imagen a la derecha e izquierda con las flechas de derecha e izquierda, y finalmente, el zoom in y zoom out con el scrollwheel del mouse.
- **WebClient.cs:** Este lee la petición que obtuvo del servidor tipo .json y manda las variables al MapGenerator para así poder inicializar el mapa de acuerdo a los valores dados por la simulación en Python step con step.

- **MapGenerator.cs:** Este inicializa las celdas (Tile), con sus respectivos objetos arriba de esta (si es que tiene).

Con esto, y el código del servidor llamado tc2008B_server.py, podemos hacer una conexión donde el servidor toma el mapa generado en la simulación cleaningRobots.py, manda los datos de este mapa al cliente en un .json, el cliente lo lee y pasa las variables para inicializar el mapa en MapGenerator.cs.

Con todo esto, el mapa en Unity replica lo que hace el matplotlib de Python, pero este lo hace en una vista 3D y con los objetos "moviéndose".

Y ¡listo! así tenemos la simulación completa del robot limpiador de basura, tanto en Python con Matplotlib como en 3D en Unity.

Link al video:

Input1: https://youtu.be/pOWeqKBKHBQ

Link al repositorio en GitHub:

https://github.com/Ian326/TC2008B_Multiagent_cleaning_robots

Referencias

Banco Mundial. (2018). *Los desechos: un análisis actualizado del futuro de la gestión de los desechos sólidos.* Recuperado el 06 de septiembre del 2023 de: https://www.bancomundial.org/es/news/immersive-story/2018/09/20/what-a-waste-an-updated-look-into-the-future-of-solid-waste-management