Actividad integradora

Daniel Isaac Ruiz Cruz - A01652366
a0652366@tec.mx

Modelación de sistemas multiagentes con gráficas computacionales

Profesores:
Sergio Ruiz Loza
David Christopher Balderas Silva

Fecha de entrega: 23 de noviembre del 2021

Parte 1: Sistemas multiagentes

Descripción del problema:

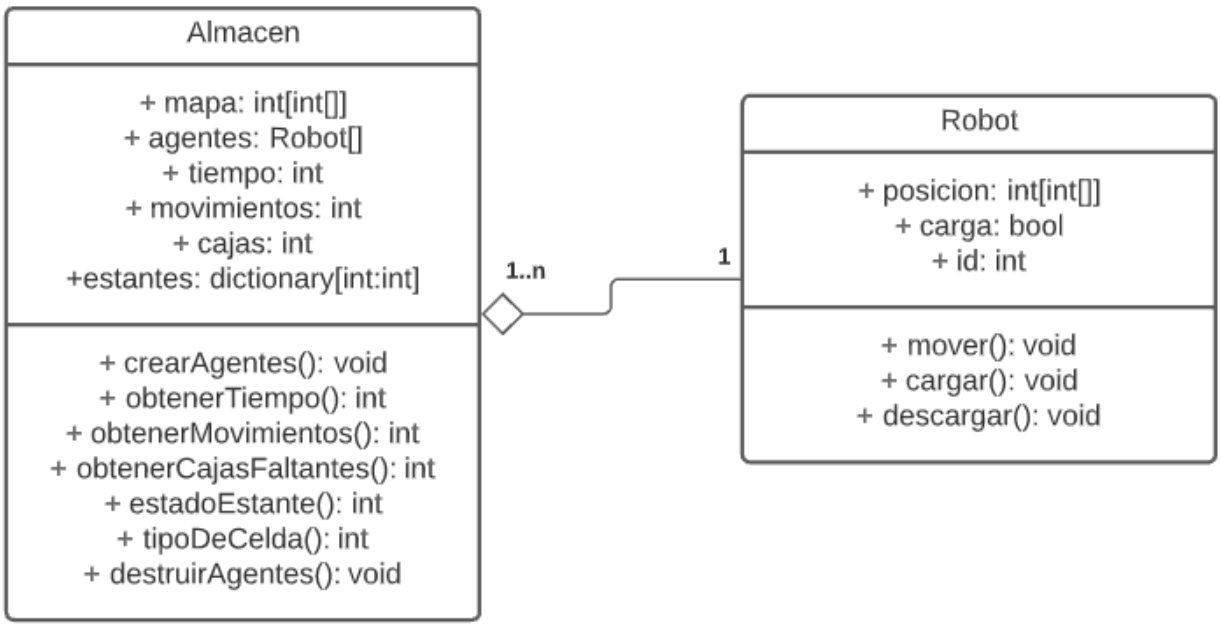
Nos indican que tenemos un almacén de tamaño NxM donde tenemos que organizar cajas esparcidas aleatoriamente con ayuda de 5 robots. La tarea es diseñar un programa que controle de manera eficiente a los robots para que logren ordenar el almacén. Para esto, se describirán los pasos a seguir, posteriormente, que guiarán a una solución fructífera y eficiente a este problema.

Estrategia de Solución:

Para implementar una estrategia, primero es necesario establecer las funciones y restricciones que sabemos. Primero, podemos modelar el almacén como una matriz de NxM donde cada celda es un espacio que puede contener: un robot, una caja o un estante. Esta matriz sería nuestro ambiente, donde los agentes tendrán que realizar sus funciones dentro de los límites establecidos (las paredes corresponderían a los límites de la matriz).

Posteriormente, se determina que los robots o agentes, se pueden mover en cualquier dirección, cargar una caja o apilarla en un estante, el cual se puede representar como un nuevo objeto "Robot" el cual tenga como métodos el moverse, apilar o cargar, de tal manera que pueda interactuar con el ambiente. Sus atributos pueden ser: posición, carga y número de identificación. Estos nos servirán para tener un mejor control de los agentes en todo momento.

Diagrama de clases:

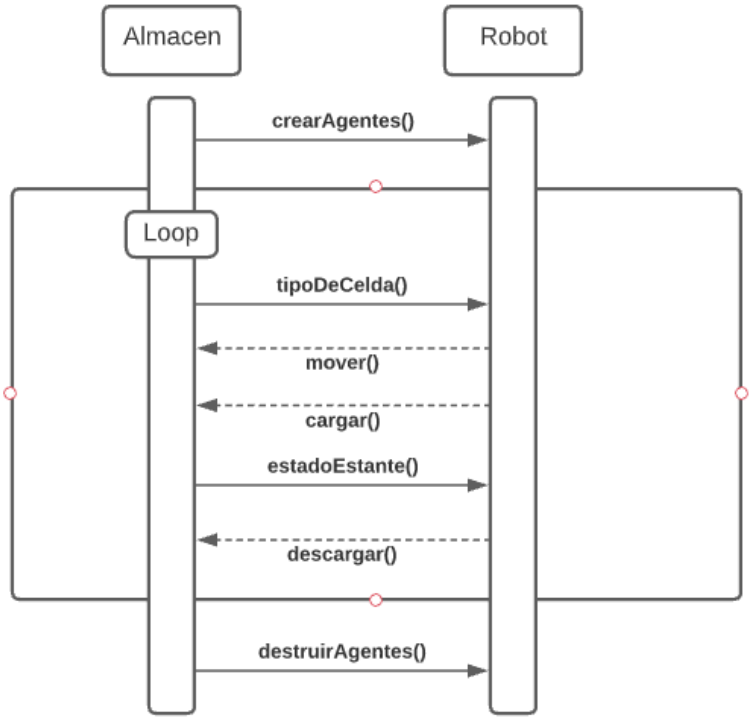


Teniendo en mente el diagrama de clases, podemos observar que la clase "Almacen" corresponde al ambiente controlado, el cual tiene un mapa como atributo que es una matriz NxM donde cada celda tiene un numero que representa un agente,

una caja, un estante o un pasillo. Asimismo, tiene parámetros de control como tiempo y movimientos los cuales ayudaran para poder saber en que tiempo y en cuantos movimientos totales por parte de los agentes se pudo ordenar dicho almacén. Aunado a ello, cuenta con una lista de agentes u objetivos tipo “Robot” que servirá para incluirlos dentro del ambiente. También, como atributo un diccionario que contiene los distintos estantes con su capacidad máxima, de tal manera que los agentes puedan preguntarle al ambiente si pueden apilar una nueva caja en el estante o tienen que buscar uno diferente. Hablando específicamente de los métodos, cada uno tiene funciones de respuesta para los agentes y para obtener los resultados finales de la operación. Cada una de las funciones será explicada posteriormente con su utilización en el diagrama de protocolo de agentes.

Por otro lado, la clase “Robot” cuenta con un atributo de posición que es una lista de dos números (columna y fila), una variable que indica si el agente esta cargando una caja y un id único para identificarlo. Sus métodos son bastante sencillos, los cuales corresponden a las distintas acciones que pueden realizar como moverse, cargar una caja o apilarla en un estante. Cabe destacar que esta clase depende totalmente del ambiente, por lo que si no hay un ambiente o un objeto tipo “Almacen” no pueden existir agentes u objetos tipo Robot.

Protocolo de agentes:



Este diagrama de protocolo de agentes sirve para identificar de una mejor manera cual sería el proceso de acción de este programa. En primera instancia, el objeto “Almacen” se crea, este a su vez crea 5 objetos tipo “Robot” que agrega a su lista de agentes. Posteriormente se inicia un ciclo, donde se checa el número de cajas

que quedan sin ordenar para saber si se ha terminado el trabajo y así destruir a los agentes o proseguir con el siguiente paso. El siguiente paso contiene 2 estados: pregunta y acción. El agente pregunta al ambiente en qué tipo de celda se encuentra, de tal manera que el agente pueda decidir si moverse, cargar una caja o apilarla. En este paso se corrobora si el agente lleva una caja, si requiere moverse o si se tiene que apilar una caja. En el caso de que el agente decida moverse, este le notifica al ambiente para actualizar su matriz y también se actualiza su posición. En caso de tener una caja al alcance, el Robot implementa el método de "cargar()" y le notifica al ambiente para actualizar la matriz. Para el último caso en que el agente tenga una caja y este al lado de un estante, este le preguntará al ambiente el estado del estante al ambiente, de tal manera de que el agente pueda saber si es posible apilar la caja o necesita buscar otro estante. Si el estante esta libre, el agente descargara la caja notificándole al ambiente para que este actualice su matriz.

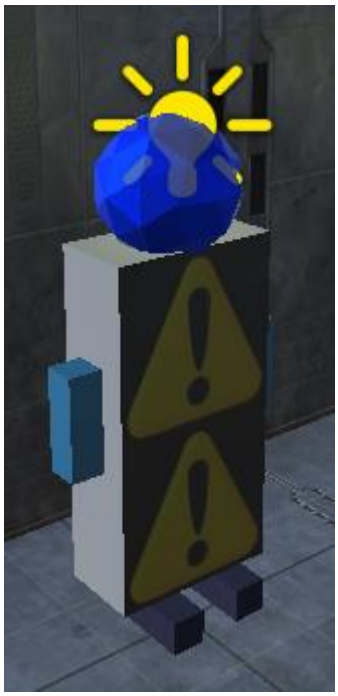
Finalmente, cuando la función "obtenerCajasFaltantes()" (obtiene el número de cajas desacomodadas) de un resultado de cero, el ciclo terminará y destruirá a los agentes del ambiente, además de imprimir en consola el resultado del tiempo y movimientos con las funciones "obtenerTiempo()" y "obtenerMovimientos()", respectivamente. Es necesario mencionar que el tiempo corresponde a un ciclo de iteración, es decir, cuando todos los agentes hayan realizado algún movimiento, además de que los movimientos son contados cada que los agentes responden al ambiente moviéndose de casilla para actualizar la matriz.

Conclusiones:

Con esta implementación hay que tener en cuenta que los movimientos de los agentes son de manera aleatoria hasta que encuentran una caja, en el momento que encuentran una caja proceden a buscar la ruta mas optima al estante mas cercano. Este método hace que el tiempo y el numero de movimientos requeridos para acomodar todas las cajas utilizando solo 5 agentes sea muy variable, pero en promedio alta. Si se busca hacer mas eficiente esto, se requeriría implementar un patrón de movimiento, donde cada agente analice una zona especifica del almacén dependiendo de la posición inicial en la que se encuentra. Esto funcionaria como la limitación de un área dentro del ambiente para cada agente, de tal manera que cada uno pueda acomodar su zona sin chocar o estorbar a otros agentes. Con esto, se reduciría el numero de movimientos por agente, puesto que tienen una ruta y camino definido, en comparación con el movimiento aleatorio que recae en la probabilidad de que en su camino se encuentren con una caja.

Parte 2: Gráficas computacionales

Para el desarrollo de esta etapa de la actividad integradora se utilizo el programa Unity con la herramienta "Pro Builder" para diseñar la escena y los agentes. En este caso, los materiales y modelos fueron obtenidos de la *Asset Store* de Unity (Los créditos de los autores están al final de este documento), con excepción del modelo del agente el cual fue diseñado a partir de figuras geométricas y decoradora con la herramienta "Pro Builder".



Por otro lado, el movimiento de los agentes fue hecho con ayuda de la función “transform” incluida en la librería de Unity. Esta función ayuda a modificar la posición de los objetos de una manera mas fluida y fácil.

```
void Update()
{
    for(int i = 0; i < robots.Length; i++ )
    {
        robots[i].transform.Translate(Vector3.right * 0.05f);
    }
}
```

Finalmente, al completar esta parte de la actividad integradora, pude incorporar los conocimientos adquiridos durante las ultimas 4 semanas del curso. Pude diseñar un nuevo objeto desde cero con “Pro builder” y darle los materiales específicos que ayudaran con su diseño.

Créditos

- Sci-Fi Construction Kit (Modular): desarrollado por Sickhead Games.
- Robot JR-1 (animated) + mod1 & mod2: desarrollado por Jope Anti-Studio.