ACTIVIDAD INTEGRADORA

1 de Diciembre del 2021

TC2008B.1

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 1)

Docentes de la materia:
Luciano García Bañuelos
José Eduardo Ferrer Cruz

Alumno:
Aarón Cortés García A01730451

INTRODUCCIÓN.

A través de este breve reporte, se expone la solución usada para la Actividad Integradora del bloque. Por medio de la teoría de multiagentes computacionales aplicada en el *framework* Mesa para Python, se solicitó crear un modelo que simulara el movimiento de robots de carga, con el fin de desplazarse dentro de un almacén e ir apilando cajas en el menor tiempo posible. Asimismo, recurriendo al motor de videojuegos Unity y su herramienta llamada ProBuilder, se pidió la representación gráfica de los componentes, buscando hacer más evidente el comportamiento y funcionamiento de los agentes.

DESARROLLO.

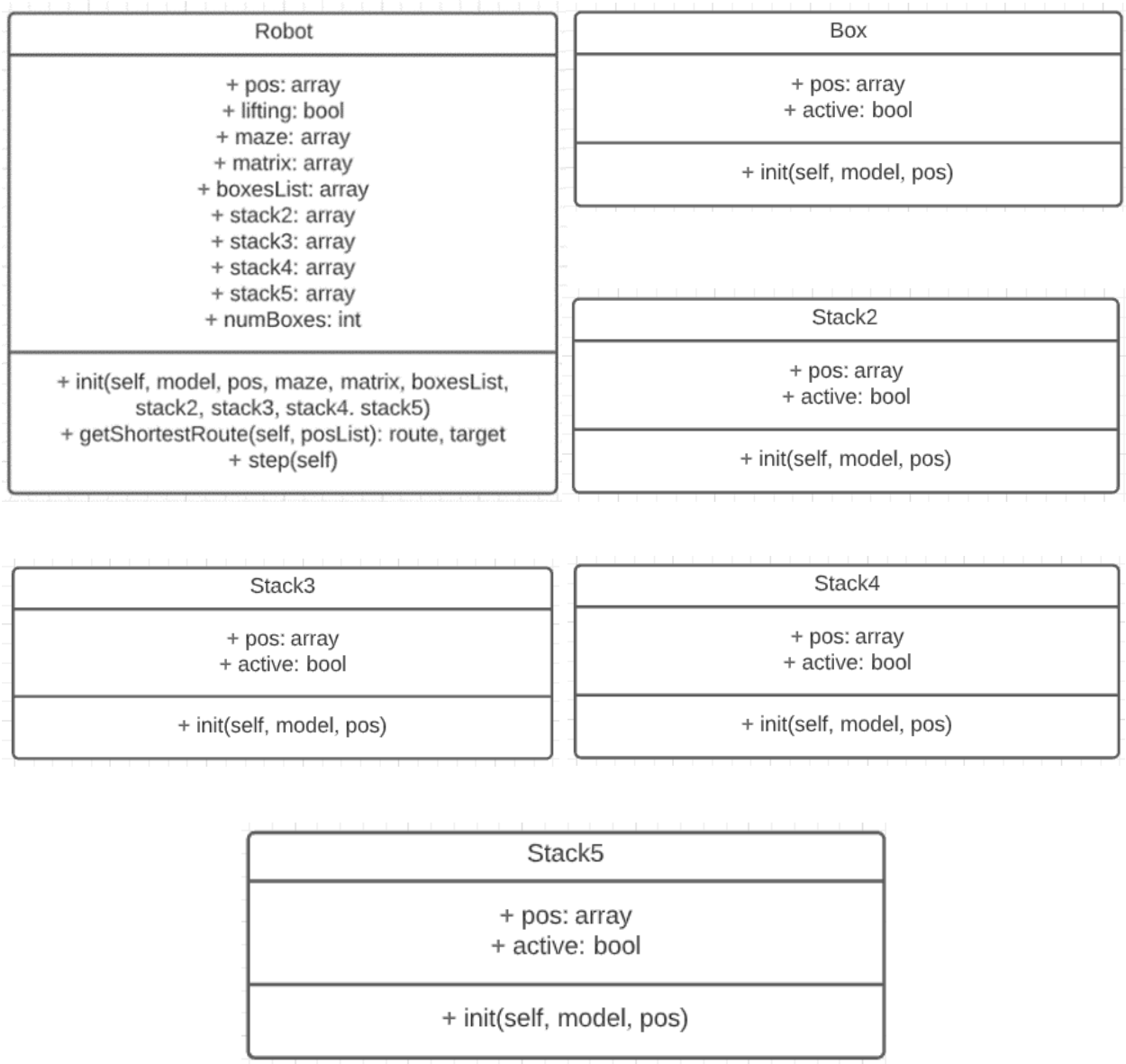
Para esta Actividad Integradora, se nos situó en el siguiente contexto: somos propietarios de 5 robots de carga nuevos, así como un almacén lleno de cajas, sin embargo, dicho almacén se encuentra en completo desorden, por lo que, utilizando los conocimientos adquiridos durante las 5 semanas del curso, se nos pide que diseñemos un modelo de multiagentes, permitiendo que los robots transporten las cajas y vayan siendo apiladas en *stacks* de máximo 5 unidades, desde luego, buscando que el tiempo para completar dicha tarea sea el menor posible.

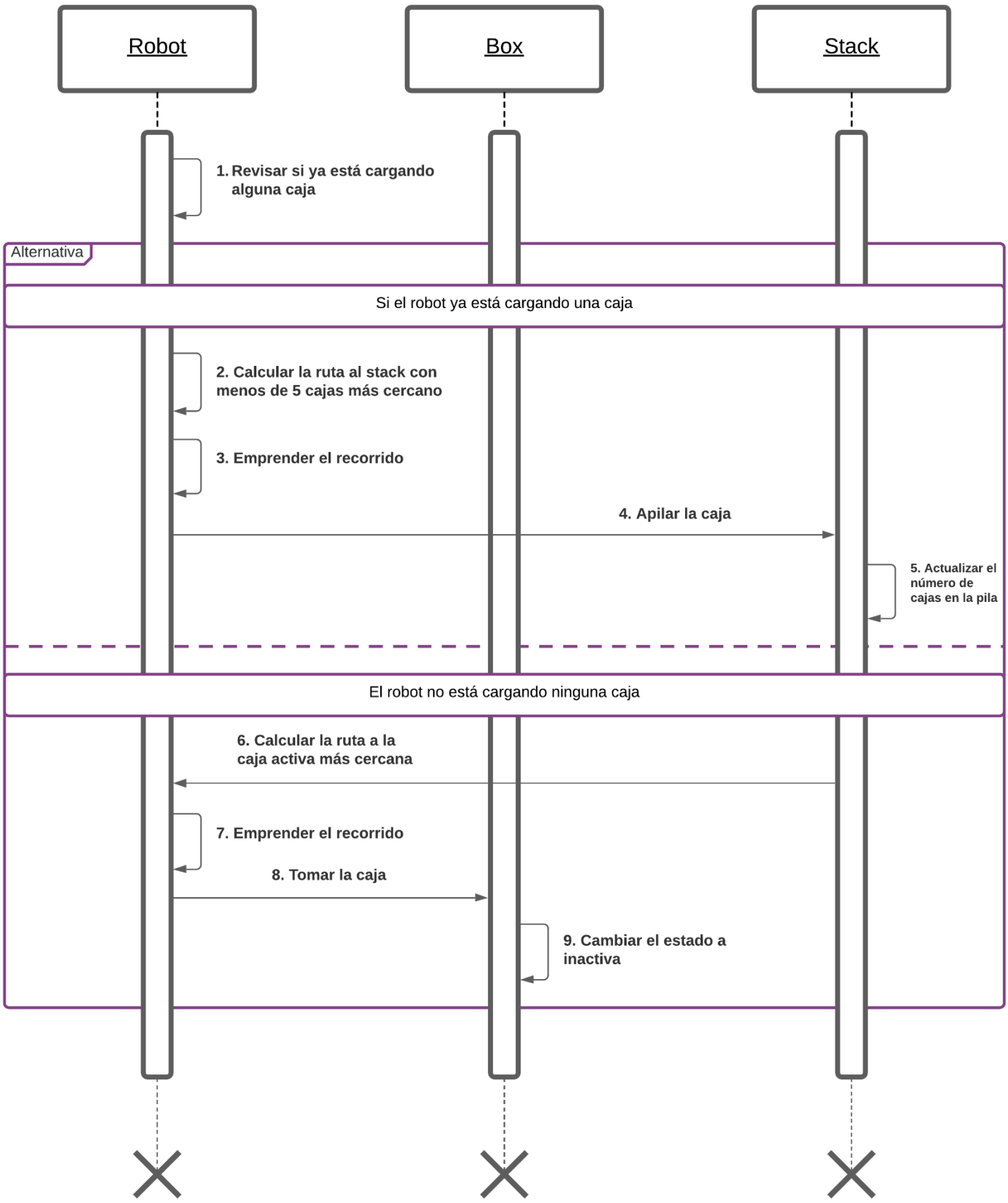
Personalmente, diseñé un modelo de agentes reactivos simples y tomando ciertos elementos de los basados en modelos. Para poder ir apilando las cajas, los robots tienen conocimiento del estado de otros agentes almacenados en listas, como por ejemplo, las cajas solitarias actualmente activas (es decir aquellas que aún no han sido recogidas del suelo) y las ubicaciones al momento de los stacks con 2, 3, 4 y 5 cajas. Además, gracias al uso de algoritmos como el A* de la librería Pathfinding, es posible obtener las rutas más cortas dentro de la cuadrícula para llegar a los destinos marcados, reduciendo de este modo el tiempo de llegada. Los robots de carga, a pesar de no tener una comunicación o interacción directa entre ellos, las modificaciones de estado que hace alguno en el almacén, son reconocidas por todos los demás, esto evita que, por ejemplo, un robot intente dirigirse a una caja que ya ha sido recogida por alguien más pasos atrás en la simulación, o que trate de dejar su caja en un stack que ya ha llegado al límite permitido de 5 unidades.

Como la colocación inicial de las cajas y los robots se hace de forma aleatoria, los resultados obtenidos varían entre simulación y simulación aún cuando los parámetros de entrada son iguales. Y aunque esto mismo evita ver un comportamiento 100% lineal al incrementar al doble o al triple el número de cajas, obviamente se reflejan cambios importantes en el número total de movimientos hechos por los robots para completar la tarea. Cabe destacar que cada robot solo suma al contador de movimientos cuando se mueve, toma o suelta una caja. A continuación se muestra una tabla comparativa con diferentes ejecuciones del programa, todas con el mismo tamaño del almacén, 20 x 20 casillas:

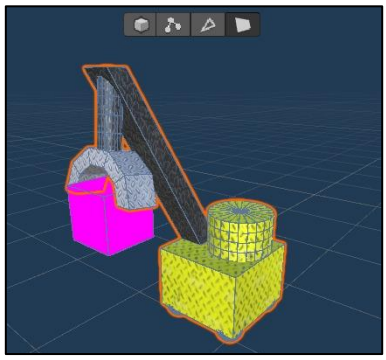
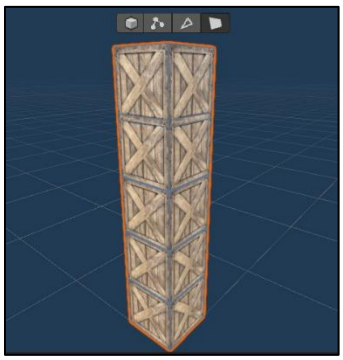
Número de simulación	% de casillas con caja	Pasos totales de la ejecución	Movimientos hechos por todos los robots
1	10%	139	644
2	10%	130	625
3	10%	126	599
4	20%	197	942
5	20%	202	960
6	20%	193	933
7	40%	332	1623
8	40%	298	1421
9	40%	355	1732

Ahora, se presentan los diagramas de clases para los agentes y el diagrama de secuencia, que muestra a nivel conceptual el protocolo de interacción:




Robot
Box
Stack
1. Revisar si ya está cargando alguna caja
Alternativa
Si el robot ya está cargando una caja
2. Calcular la ruta al stack con menos de 5 cajas más cercano
3. Emprender el recorrido
4. Apilar la caja
5. Actualizar el número de cajas en la pila
El robot no está cargando ninguna caja
6. Calcular la ruta a la caja activa más cercana
7. Emprender el recorrido
8. Tomar la caja
9. Cambiar el estado a inactiva

Para la parte de la representación gráfica en Unity, simplemente hice unos modelos básicos con ProBuilder, tomando figuras geométricas, modificando sus dimensiones, caras y vértices para darles la forma de los agentes. Luego, con la creación de materiales basada en imágenes y el editor UV, pude agregar texturas al modelo del robot, la caja y el almacén en sí. Todos estos ejemplares los guardé como Prefabs, para que posteriormente solo se hicieran instancias de los objetos con las mismas propiedades.

Para conectar la parte del modelo hecha en Mesa y la parte gráfica en Unity, tuve que escribir un script en Python para lanzar una aplicación con Flask, lista para responder a solicitudes. Con el método POST, se crea una nueva simulación y se retorna la referencia a esa instancia. El método GET avanza un paso en la simulación y retorna un JSON con todos los datos que necesita recibir Unity, como la posición de los agentes y si las cajas están activas o no. Finalmente, creé otro script, pero ahora en C#, para que una vez corriendo el backend hecho en Python, se pueda interactuar con la simulación, logrando obtener los datos relevantes, instanciar los GameObjects de los Prefabs e irlos moviendo o desapareciendo según corresponda.

CONCLUSIÓN.

A manera de conclusión, me gustaría responder a la pregunta, **¿como se podría optimizar todavía más el tiempo en que los robots completan la tarea? Y, de manera general, ¿cómo podría mejorarse el código?**

En este momento, cada robot va en busca de la caja más cercana a él, y si ya se encuentra cargando una, revisa la existencia de stacks en el almacén para apilarla allí. Pero esto último lo hace en un orden particular, de mayor a menor número de unidades en los stacks. Es decir, primero busca un stack de 4 cajas, si no hay, busca uno de 3, y así sucesivamente, por lo que, aún cuando un stack de 2 cajas esté justo a lado del robot y lo más conveniente sea colocar su caja en esa pila, preferirá desplazarse hacia una con más cajas, ignorando si está más lejos. Por lo tanto, quizá una buena idea sería segmentar la cuadrícula del almacén en 5 zonas diferentes, para luego asignarle una a cada robot y reducir su área de desplazamiento, dando paso a una mayor organización y eficiencia. Adicionalmente, podría establecerse una comunicación

continua entre robots, para que, cuando uno ya haya terminado de apilar las cajas en su zona, reconozca qué robot es el que necesita más ayuda y vaya a apoyarlo. Al hacer un análisis del funcionamiento, veo que probablemente puedan evitarse tener muchas variantes del agente stack, y solo tener uno solo, pero con un atributo que cuente las unidades apiladas. Y por último, para hacer una simulación más realista, se podría optimizar el movimiento de los robots, para que eviten ponerse en un mismo lugar, traspasar cajas o stacks, pues esto en un contexto verdadero no es válido.

Por otro lado, la representación gráfica en Unity tiene mucho margen de mejora, comenzando con los modelos en sí, que podrían hacerse más detallados y visualmente atractivos. Asimismo, podrían aplicarse objetos y sistemas de iluminación o activar las colisiones entre GameObjects, presentando un movimiento más fluido y con aplicación de físicas.