Actividad Integradora

Modelación de sistemas multiagentes con gráficas computacionales

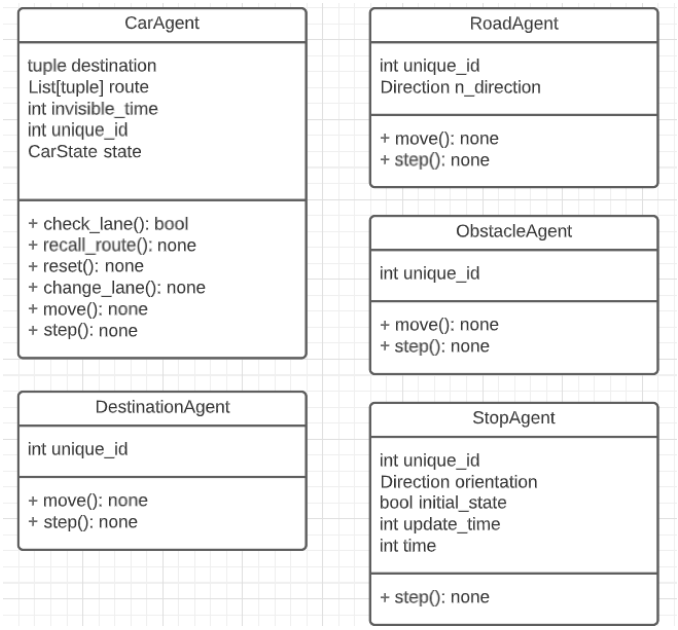
Stephan Guingor Falcon A01029421
Salvador Salgado Normandia A01422874

11/31/22

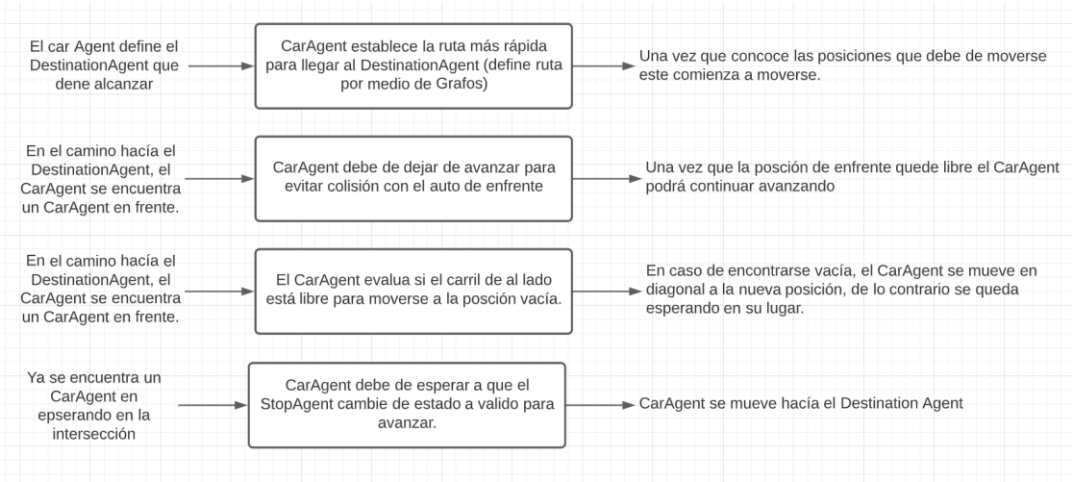
Revisión 3 - Avance al 60%

Continuando con la revisión 2 (Modelación de agentes), fue momento de llevar a cabo la implementación de un cruce con semáforo para un modelo de una ciudad completa. En este caso tuvimos que hacer varias modificaciones a nuestros agentes iniciales, ya que en dicha revisión se consideraba que las intersecciones se guiaban por medio de señales de alto. De igual manera tuvimos que evaluar que el diseño del grid donde los agentes interactuaban era creado a partir de un archivo txt, por lo que era necesario implementar dentro del código tanto la lectura como la generación de los agentes en su posición correspondiente. Antes de entrar en detalles sobre los avances de la entrega es necesario mostrar los nuevos diagramas de clases para cada agente.

Diagramas de Clases



Protocolo de Agentes



Para este modelo consideramos que además del CarAgent presentado en la actividad anterior sería necesario considerar que para la lectura del txt con el diseño de la ciudad sería más efectivo considerar cada carácter como un posible agente. Con esto dicho se puede observar que exceptuando al CarAgent, el resto no cuentan con tanta complejidad. El único que llega a ser más avanzado vendría siendo StopAgent, el cual además de proporcionar la dirección hacía donde debe moverse al auto también se encarga de avisar al CarAgent cuando debe de frenar según su initial_state.

Plan de trabajo y aprendizaje Adquirido

Trabajo Actualizado

Desarrollo de Agentes y Modelo

A diferencia de las entregas pasadas donde nos basábamos en los templates del profesor, está vez decidimos diseñas tanto el archivo de agentes como modelo desde cero, lo cual nos facilitó al momento de implementar nuestro algoritmo de grafos al CarAgent para encontrar la mejor ruta posible. Se podría decir que durante este desarrollo se fue la mayoría del tiempo en el diseño de los grafos, ya que era necesario considerar las diversas direcciones posibles (“>”, “<”, “v”, “^”) para llegar al DestinationAgent. Al final pudimos asegurarnos de cumplir con estos limitantes al igual que permitir que la ruta considerada el movimiento entre carriles.

Estimación inicial para completar: 8 horas
Tiempo real: 12 horas
Diferencia: +4 horas

Desarrollo de API para comunicarse con Unity

En este caso la creación del Script AgentController en Unity resultó muy similar al resto de actividades, realizando las llamadas a nuestro Api para obtener la información de todos los agentes en cada step realizado. De igual manera este AgentController contaba con los prefabs de cada uno de los agentes, los cuales cuentan con varios tipos de modelos que pueden usar. Un tema necesario para destacar fue la creación de un nuevo Script llamad CarController, el cual es el encargado de hacer que el movimiento del CarAgent sea lo más realista posible por medio se realizar en cambio de posición por medio de rotaciones que simulan el movimiento al hacer curvas o cambiar de carril.

Estimación inicial para completar: 4 horas
Tiempo real: 6 horas
Diferencia: +2 horas

Creación de modelos en Blender

Durante esta actividad fue necesario diseñar una ciudad entera con sus respectivas intersecciones ya avenidas. Para este caso nos dimos a la tarea de diseñar los siguientes modelos en Blender:

Vehículos - CarAgent’s (con distintas texturas)
Calles de la ciudad – RoadAgent
Edificios de distas formas y tamaños – ObstacleAgent

Semáforos – StopAgent

Es importante destacar que tanto los Prefabs como Texturas usadas en este proyecto fueron realizado por nosotros.

Estimación inicial para completar: 7 horas
Tiempo real: 9 horas
Diferencia: +2 horas

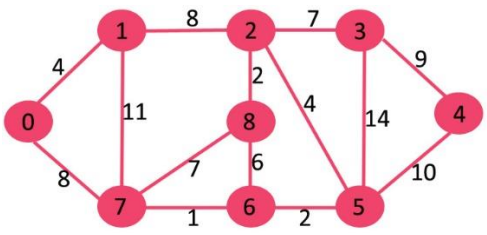
Implementación Gráfica Unity

En cuestiones Gráficas de Unity tuvimos algunas complicaciones al momento de definir la correcta posición de algunos Prefabs según la dirección del agente. En el caso de los StopAgents y RoadAgents fue necesario modificar la rotación según la dirección que se obtenía por medio de la llamada al Api. Otro tema importante para destacar fue la corrección de tamaños de los diversos Prefabs, ya que al ser exportados de Blender estos no contaban con escalas similares.

Estimación inicial para completar: 6 horas
Tiempo real: 7 horas
Diferencia: +1 horas

Aprendizaje adquirido como equipo

Durante este avance del proyecto nos resulto muy beneficioso el optar por hacer el modelo desde cero, aún cuando ya se tenía una base brindada por el profesor. Esto nos permitió re investigar conceptos de Python vistos previamente, como lo fue el caso de lectura de archivo txt, uso de funciones lambda al igual que poner en práctica las comprehension list vista en clase del profesor Octavio. De igual manera nos resulto de gran ayuda el primero desplegar nuestro modelo en el servidor básico de mesa antes de pasar a desplegarlo en Unity. De esta manera nos pudimos asegurar que tanto el grafo implementado para detectar el camino más corto al igual que el correcto funcionamiento de los agentes estuvieran funcionando correctamente. En el tema de definir la mejor ruta a los DestinationAgents nos permitió investigar e intentar implementar diversos algoritmos de gráficos, lo cual consistió en un gran reto al tratarse un mapa el cual contaba con un sin de posibles rutas, al igual que era necesario considerar limitante como moverse a celdas con dirección incorrecta o la posibilidad de cambiarse de carril. Al final pudimos implementar el algoritmo Dijkstra, el cual nos otorgaba una lista de coordenadas que se debían de tomar para llegar al objetivo. Una de las ventajas de este approach es que a medida que la simulación se va realizando se van detectando mejores caminos, por lo que el tiempo de llega disminuye en gran medida.



Actividades Pendientes

Ahora que hemos solucionado el tema de colisiones de vehículos al igual que renderado correcto de cada agente dentro de Unity, es necesario utilizar IBM Cloud para poder instanciar nuestro servidor donde habitara el modelo. De esta forma sólo será necesario configurar el script de Unity para que haga las llamadas API a la nube y obtener las nuevas posiciones de los agentes. De igual manera consideramos que la escena en Unity puede mejorarse visualmente, por lo que estaremos implementando nuevos modelos de Blender como lo pueden ser arboles alrededor del mapa, así como mejores azoteas, luces de vehículos al igual que cambio de luces en los semáforos.

Creación de nuevos modelados en Blender
Participante Encargado: Salvador Salgado Normandia
Fecha en la que se realizará: martes 29 de noviembre
Esfuerzo Estimado (horas): 4 horas

Desplegar aplicación en IBM Cloud
Participante Encargado: Stephan Guingor Falcon
Fecha en la que se realizará: martes 29 de noviembre
Esfuerzo Estimado (horas): 2 horas

Revisión 4 - Avance al 100%

Plan de trabajo y aprendizaje Adquirido

Trabajo Actualizado

Desarrollo de Agentes y Modelo

Al mostrar el avance previo en clase se nos mencionó que la manera en la estábamos manejando los CarAgents al llegar al destino, el cual simplemente los volvía a colocar en una nueva posición dentro del grid y les asignaba un nuevo DestinationAgent. En este caso no se iban generando nuevos autos, por lo que la simulación de tráfico no se podría ver en acción. Con esta nueva información optamos por hacer un rediseño en nuestro model, en donde al llegar a los DestinationAgents, los autos serían borrados por completo (según su atributo **state**). De igual manera se agregó una variable llamada frequency, el cual establece el numero de coches que se iran agregando después de cada 5 steps. De igual manera se agregó una variable llamada activation_time, la cual define el número de steps que dura cada semáforo antes de cambiar al siguiente estado.
Estimación inicial para completar: 5 horas
Tiempo real: 4 horas
Diferencia: -1 horas

Desarrollo de API para comunicarse con Unity

Una vez que implementamos las modificaciones tanto al modelo como alguno de los agentes nos dimos cuenta de que durante la simulación en Unity se podía observar que los autos no se destruían al llegar al DestinationAgent al igual que los StopAgent no cambiaban de luz al pasar el tiempo definido. En un principio no lográbamos detectar cual podía ser causante de este error, pero al entrar más en detalle pudimos darnos cuenta de que habíamos olvidado modificar nuestro código de agents.py para que este mandará estos nuevos atributos por medio de las llamadas API.

```python
@property
def serialized(self) -> dict:
    return {**super().serialized, **{
        "orientation": self.orientation,
        "active": self.active
    }}
```

Obtener variables del StopAgent con serialized()

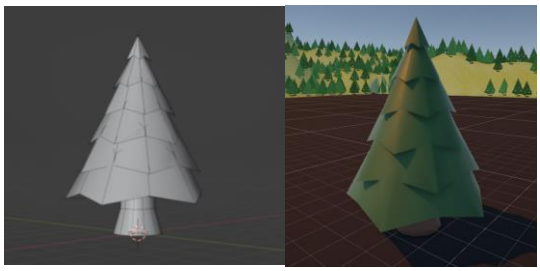
```python
@property
def serialized(self) -> dict:
    return {**super().serialized, **{
    "state": self.state,
}}
```

Obtener variable *status* de CarAgent con serialized()

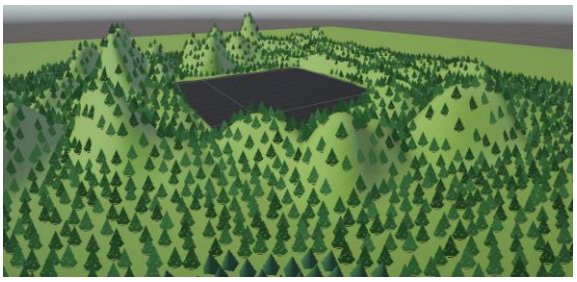
Estimación inicial para completar: 1 horas
Tiempo real: 2 horas
Diferencia: +1 horas

Creación de modelos en Blender

En la entrega anterior se había logrado diseñar los modelos primordiales para la simulación, por lo que en esta etapa nos enfocamos en añadir más detalles visuales a la simulación, en este caso consideramos que sería visualmente atractivo el colocar la ciudad en un valle, por lo que nos dimos la tarea de implementar los relieves dentro del terrain, al mismo tiempo que optamos por modelar un árbol desde cero en Blender para insértalo con la opción de insert Trees.



Proceso de creación del árbol



Visualización del terreno con relieves y árboles

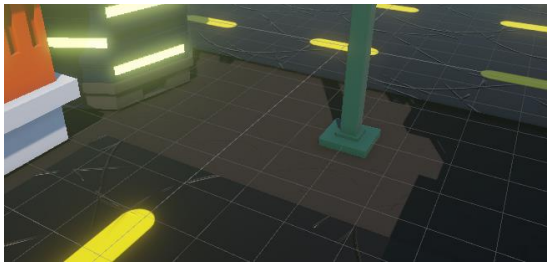
Estimación inicial para completar: 2.5 horas
Tiempo real: 1 horas
Diferencia: -1.5 horas

Implementación Gráfica Unity

Durante el comienzo de esta etapa nos percatábamos que aún contando con las coordenadas correctas definidas por el grafo, al momento que los CarAgents se movían por el grid experimentaban cambios de posiciones que no coincidían con el comportamiento mostrado en la ruta. Al entrar en más detalle a nuestro código pudimos observar que el problema radicaba dentro de la función Update() en CarController.cs, ya que la forma en la que definíamos la variable dt no coincidía con el approach usado por el profesor. Una vez que hicimos la corrección se pudo observar un comportamiento normal de los vehículos.

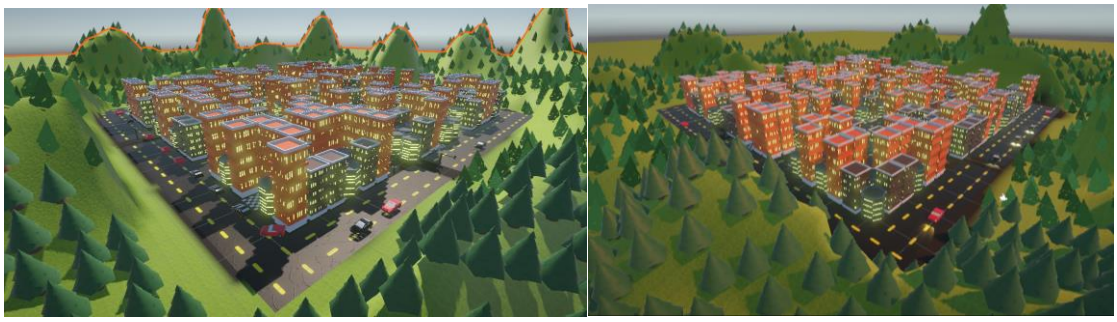
```
if (updated)
{

    timer -= Time.deltaTime;
    if (timer < readonly struct System.Int32
        timer = 0;
    }
    else{
    dt = 1.0f - (timer / timeToUpdate);

    foreach(var car in cars)
    {

       car.Value.GetComponent<CarController>().MoveTo(currPositions[car.
       Key], dt, timer);
    }
    }
}
}
```

Debido a que la mayoría de la implementación gráfica de los agentes ya había concluido en la entrega pasada, durante esta entrega se enfocó en terminar de agregar detalles adicionales a la escena. Cómo se pudo ver en la entrega anterior, tanto los StopAgents como los ObstacleAgents no contaban con un piso por default, por lo que se opto por crear un terrain debajo de la ciudad donde se pinto los huecos con una textura similar al RoadAgent.

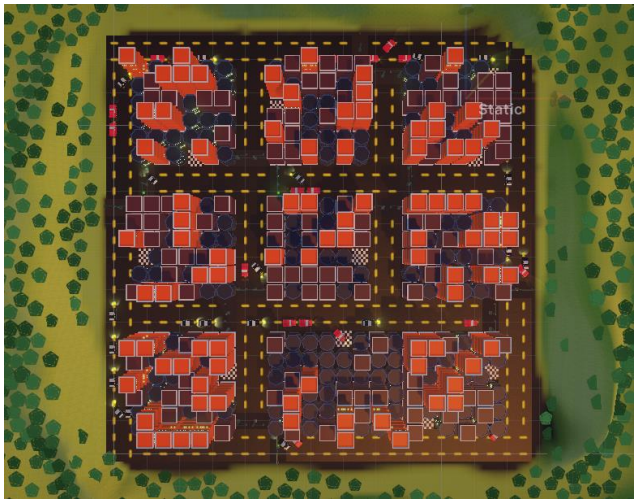


El terrain se distingue al texture RoadAgent de manera significativa, pero en aspecto general logra mantener similitud.

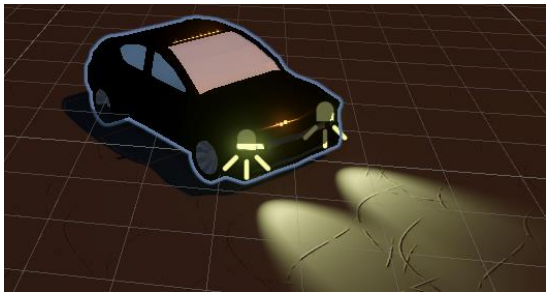
De igual se decidió incluir en los CarAgents la iluminación correspondiente a sus luces frontales para un mayor realismo. Para que se pudiera observar de mejor manera tanto la ilumnación de los autos, así como las luces de los edificios decidimos descargar un skyblock desde AssetStore para poder crear un tono de luz que simulara el atardecer o la tan famosa *Golder Hour*. De igual manera se estuvo experimentando con las opciones del terrain para generar relieves y colocar una gran cantidad de árboles.



Comparativa con el uso de Skyblock



Vista aérea de la simulación



Para el caso de la iluminación de los autos se utiliza luz tipo Spotlight, en donde fue necesario realizar ajustes tanto en el ángulo de reflexión (para que se observara en el piso), intensidad y color.

Estimación inicial para completar: 3 horas
Tiempo real: 1.5 horas
Diferencia: -1.5 horas

Aprendizaje adquirido como equipo

Durante esta etapa final pudimos ver la gran utilidad de haber diseñado los APIs de una manera correcta, ya que al hacer las peticiones desde Postman fue donde nos pudimos percatar que para cada step que se obtenían, estos no contaban con los nuevos atributos (status, active). De igual manera se pudo continuar desarrollando nuestras habilidades tanto en el uso de Blender para diseñar modelos de mejor calidad al igual que seguir implementando las diversas funcionalidades que ofrece Unity (terrain, spotlights, etc).

Consideraciones finales

Análisis de la solución

Reflexión final