Ejercicio 2D Netlogo

**1. Planteamiento del problema:**  
Se requiere crear una simulación de un sistema de entrega a domicilio para un restaurante. El restaurante está ubicado en el centro de una cuadrícula, lo que permite una distribución de pedidos relativamente uniforme alrededor de él. La cuadrícula tiene dimensiones de 45 x 45, y el repartidor se desplaza a una velocidad de 20 celdas por minuto. Esto significa que el repartidor puede recorrer desde el centro (donde se encuentra el restaurante) hasta uno de los extremos en aproximadamente un poco más de un minuto.  
  
El número de pedidos se distribuye entre casas y edificios. Cada casa tiene un promedio de 3 pedidos por minuto. Los edificios pueden tener entre 4 y 20 apartamentos (casas) cada uno. En total, hay 50 residencias (casas y edificios) distribuidas aleatoriamente en la cuadrícula. La probabilidad de que una residencia sea una casa es de 0.3, mientras que la probabilidad de que sea un edificio es de 0.7.  
  
La simulación tiene una duración de 180 minutos (3 horas) y se solicita determinar el número de repartidores necesario para garantizar que el tiempo de espera promedio por pedido sea inferior a 20 minutos.  
  
Por último, se considerará la opción de contratar bicicletas en lugar de motocicletas. Las bicicletas tienen un costo equivalente a un tercio del costo de las motocicletas iniciales, pero se desplazan a la mitad de velocidad, es decir, 10 celdas por minuto. Para esta simulación, se obtendrán los mismos resultados iniciales para determinar cuál es la mejor alternativa.  
  
  
**2. Plan general y problemas a resolver:**El enfoque principal para abordar este problema consiste en desarrollar una simulación basada en agentes, donde se consideran tres tipos de agentes: repartidores, casas y edificios. Cada uno de estos agentes tiene parámetros específicos, como el número de visitas a residencias (num-visited) y la velocidad de los repartidores.  
  
Uno de los desafíos críticos a resolver es encontrar una manera de independizar las rutas de viaje de cada uno de los repartidores, al mismo tiempo que se sincronizan las rutas en función de las residencias visitadas.  
  
  
**3. Planteamiento en Netlogo**Definimos el entorno como una cuadrícula de tamaño 45x45 en la sección de configuración (settings). A continuación, creamos tres tipos de agentes (repartidores, casas y apartamentos) y un agente adicional llamado "spawns" para realizar tareas específicas. Para lograr esto, utilizamos los "breeds" correspondientes.

Cada tipo de agente tiene sus propias propiedades o parámetros definidos de la siguiente manera:  
  
Repartidores: route, route counter, spawn-target y target.  
Casas: num-visited.  
Apartamentos: num-visited.  
Configuramos la rutina de inicio (setup) secuencialmente de la siguiente manera:  
  
clear-all: Borra todos los agentes existentes.  
create-delivers: Crea los repartidores.|  
create-houses: Crea las casas.  
create-departments: Crea los apartamentos o edificios.  
make-routes: Define las rutas para cada repartidor.  
reset-tickets: Reinicia los contadores de tiempo de simulación y pasos (ticks).  
  
En la rutina create-delivers:  
Creamos un número determinado de repartidores utilizando un control deslizante (slider) en la interfaz. Algunas de las propiedades importantes que se establecen en esta rutina son:  
  
Definimos los valores iniciales de route, target y route counter como una lista vacía, ningún agente y 0, respectivamente.  
  
Dentro de esta rutina, hacemos uso de la solicitud "ask" para los repartidores, utilizando "hatch" para crear hijos de los repartidores que heredan las mismas propiedades, pero con identificadores de who diferentes. Esto nos permite distinguir entre los diferentes repartidores según sus identificadores únicos. También utilizamos esta solicitud para establecer el spawn-target, que indica dónde debe dirigirse el repartidor una vez que llegue a su destino.  
  
En las rutinas create-houses y create-apartments:  
Combinamos estas dos rutinas debido a su implementación similar. Utilizamos la función "create-breed" donde el "breed" puede ser casas o apartamentos. Sin embargo, debemos tener en cuenta que la creación de cada agente tiene una probabilidad asociada, por lo que redondeamos los valores correspondientes. Si la creación es para una casa, redondeamos el número máximo de residencias según un factor de 0.3, de lo contrario, lo hacemos con un factor de 0.7. También definimos los colores de las casas y los apartamentos, su posición inicial de forma aleatoria y otros parámetros estéticos, como el tamaño y la forma.  
  
En la rutina make-routes:  
La idea principal de esta rutina es vincular dos listas. La primera lista corresponde a los repartidores y su cantidad se define a través del control deslizante en la interfaz. La segunda lista contiene las rutas ordenadas según el pedido. De esta manera, simulamos el orden de los pedidos. Para realizar esta coincidencia, utilizamos la solicitud "ask" para los repartidores y el ciclo "foreach". Inicialmente, establecemos la lista "devCasas-order" como una lista de 15 elementos iguales a 0. Esto nos permite asignar rutas únicamente al primer elemento de la lista de repartidores. Luego, pasamos la lista de casas ordenadas (houses) a través del ciclo "foreach". Para cada par de repartidor-casa, asignamos la casa correspondiente a la ruta del repartidor utilizando la función "lput". Esto asegura que cada repartidor tenga su propia lista de rutas. Si se incluyen edificios, se realiza un proceso similar, ajustando el número de elementos en la lista "devCasas-order" a aproximadamente 35.  
  
En la rutina go:  
Ejecutamos la simulación utilizando la instrucción "go". Básicamente, definimos la propiedad "target" del repartidor utilizando la solicitud "ask". Si el repartidor aún no tiene un objetivo asignado, buscamos en la lista de rutas para encontrarlo, lo cual fue establecido previamente en la rutina make-routes. Si no se encuentra un objetivo, significa que todas las residencias ya han sido atendidas o no hay pedidos pendientes. Para mover a los repartidores, utilizamos la simulación continua de la interfaz, que se activa al presionar el botón "go". Dentro de la rutina "go", verificamos si el repartidor ha llegado a su destino. Si es así, lo dirigimos a otra residencia o al "spawn" según corresponda, utilizando las palabras clave "fd", "face" y "set". Siempre que finaliza la rutina "go" y el repartidor no se dirige al "spawn", establecemos el objetivo como "nobody". Esto nos permite identificar que el repartidor debe realizar otra entrega cuando ingrese nuevamente a la rutina "go" y se dirija a "nobody". Incrementamos el contador "route-counter" y obtenemos el siguiente objetivo al revisar el siguiente elemento en la lista de rutas. Si el repartidor se dirige al "spawn", simplemente incrementamos el contador "route-counter". Para evitar que la simulación se detenga abruptamente, incluimos algunas líneas finales que repiten la misma ruta anterior, ya que es probable que para el momento en que se complete, ya se hayan realizado nuevos pedidos y se deba seguir el orden anterior debido a su antigüedad.  
  
  
**4. Documentación según el protocolo ODD:**Propósito:  
El propósito de este modelo es simular y encontrar el método más eficiente para la entrega de pedidos, considerando diferentes escenarios, como que el repartidor lleve solo un pedido o varios al mismo tiempo, o si se utilizan vehículos con radios más amplios o económicos. El objetivo es crear un sistema eficiente y rentable que brinde un excelente servicio a los clientes y genere altas ganancias para el restaurante.  
  
Entidades, variables de estado y escalas:  
Este modelo incluye tres agentes: repartidores, casas y apartamentos.  
  
Resumen del proceso y planificación:  
1. Se definen los "breeds" y sus respectivos parámetros.  
2. Se definen las rutinas a utilizar, como "setup" y las subrutinas internas.  
3. Se crean los agentes siguiendo las indicaciones del problema y se utilizan variables dinámicas (sliders) en la interfaz.  
4. Se define el agente auxiliar "spawn" para distinguir los repartidores y conocer su posición en todo momento.  
5. Se establecen detalles estéticos, como colores, formas y tamaños, en las creaciones de los agentes.  
6. Se realiza la asignación de rutas a cada repartidor según las residencias correspondientes (casas o edificios), utilizando propiedades internas definidas previamente.  
7. Se inicia la simulación continua mediante el botón "go" y se establecen las condiciones para los movimientos de los repartidores mediante un bucle "while". Aquí se define la dirección y continuidad de los destinos de cada repartidor.  
8. Se realizan las estadísticas necesarias para analizar los resultados.  
  
Inicialización:  
La simulación se inicializa con las siguientes rutinas:  
- "clear-all": Borra todos los agentes existentes.  
- "create-delivers": Crea todos los repartidores.  
- "create-houses": Crea las casas.  
- "create-departments": Crea los apartamentos o edificios.  
- "make-routes": Define las rutas para cada repartidor.  
- "reset-tickets": Reinicia los contadores, como el tiempo de simulación y los pasos (ticks).  
  
  
**5. Resultados**Se realizó una simulación del modelo durante 180 minutos, aunque por defecto se establece como una simulación infinita para determinar los tiempos de espera promedio por pedido. La simulación se llevó a cabo utilizando un único repartidor para extrapolar los tiempos de múltiples repartidores.  
  
Se observó que un repartidor en motocicleta, con una velocidad de 20 celdas por minuto, atiende en promedio una casa cada 52 segundos. Esto se debe a que cada vez que realiza un pedido, debe regresar al restaurante, lo cual representa en promedio un desplazamiento de 20 celdas. Sin embargo, si el repartidor puede realizar dos pedidos simultáneamente, el tiempo entre pedidos se reduce a aproximadamente 38 segundos.  
  
Con base en estos datos y mediante análisis y simulación, se determinó que se necesitan aproximadamente 30 repartidores para atender tanto a los edificios como a las casas en menos de 20 minutos. Es importante tener en cuenta que para los edificios se requieren múltiples vueltas debido al número de pedidos que se reciben.  
  
Por otro lado, al cambiar a bicicletas, se observó que el tiempo entre pedidos aumenta a aproximadamente 1 minuto y 10 segundos. Esto implica que se necesitarían alrededor de 48 repartidores en total para lograr los resultados deseados.  
  
  
**6. Análisis**Si consideramos que la compra de bicicletas representa solo una tercera parte del costo de las motocicletas, podemos concluir que es una opción rentable. Esto se debe a que la demora en la entrega de pedidos no sigue una relación lineal, es decir, la demora de las bicicletas no es tres veces mayor que la de las motocicletas.  
  
Además, existen otras variables que vale la pena considerar. Por ejemplo, el tráfico puede tener un impacto significativo en los tiempos de entrega, por lo que sería necesario tener en cuenta este factor al tomar decisiones. Asimismo, sería útil utilizar distribuciones más realistas de pedidos, considerando patrones de demanda y ubicaciones específicas de los clientes.  
  
También podríamos evaluar la apertura de sucursales adicionales o reconsiderar la ubicación del restaurante actual para optimizar los tiempos de entrega. Por último, definir los puntos de spawn de los repartidores de manera distribuida en la cuadrícula podría reducir los tiempos de espera de manera significativa.  
  
Estas variables adicionales y consideraciones estratégicas pueden contribuir a mejorar aún más la eficiencia y rentabilidad del sistema de entrega de pedidos del restaurante.  
  
  
**7. Dificultades**  
En el desarrollo del modelo, surgieron algunos inconvenientes relacionados con la coordinación de los repartidores al momento de realizar múltiples entregas y al dirigirse a los departamentos en lugar de casas individuales. Otro problema identificado fue la necesidad de que los repartidores regresaran al restaurante para reabastecerse antes de continuar con la ruta de entrega. Estos desafíos se presentaron durante la simulación y requirieron atención y soluciones adecuadas.