Lab 01: Implementing a Class

UPF-POO22-G202-21

Ariadna Prat / Chris Matienzo

5. Documentation

5.1 El programa consiste en simular el movimiento de un número de circulos (`numAgents 10`), llamados agentes (`public class Agent`), en un canvas determinado (`ancho: 800, alto: 600`). Estos agentes se mueven dentro del canvas hasta llegar a su objetivo, el cual es un vector de posición generado aleatoriamente (`x, y`), una vez alcanzado el objetivo cambia de dirección hacia el siguiente objetivo.

Las clases definidas son:

- `public class Agent`
  - `public Agent(Vec2D pos, double radius)` // Es el método constructor que inicializa el vector de posición y un radio para los agentes
  - `public void setTarget(Vec2D target)` // Método setter que posiciona el objetivo en el canvas
  - `public void setSpeed(double speed)` // Método setter que establece una velocidad para el agente
  - `public Vec2D updatePosition()` // Actualiza la posición del agente para que avance desde su posicion hacia una dirección y a una cierta velocidad.
  - `public boolean targetReached()` // Método booleano que verifica si el agente ha alcanzado su objetivo
  - `public boolean isColliding(Agent agent)` // Método booleano que verifica si ha habido colisión entre dos agentes
  - `public void agentPaint(Graphics g)` // Crea visualmente el agente con unas características dadas

- `public class World`

- `private Vec2D randomPos()` // Metódo privado que devuelve un vector de posición con valores aleatorios
- `private double randomRadius()` // Método privado que devuelve un radio entre 5 y 30
- `public World(int widht, int height, int margin)` // Método constructor que inicializa la altura, ancho y margen del canvas
- `public void simulationStep()` // Verifica si el agente ha llegado a su objetivo, si es así, entonces cambia de dirección hacia un nuevo objetivo
- `public void manageCollisions()` // Verifica si hay colision entre los mimos agentes, si es así, entonces cambia de dirección hacia un nuevo objetivo
- `public void paint(Graphics g)` // Crea visualmente los 10 agentes en el canvas

5.2 El método del paint que estaba escrito en el ejercicio no nos resultaba muy práctico para implementar en el código porque pedía llamar a atributos de la clase Agent dentro de la clase world. Así que implementamos un método llamado `Agent.paint` para poder usar los atributos de la clase Agent y luego usar ese método dentro del método paint que está dentro de la clase world.
Otro problema que hemos tenido es con el método updatePosition() porque al principio no sabíamos cómo multiplicar una variable double con una variable de clase `Vec2D`. Por tanto, como la clase `Vec2D` tiene dos atributos de tipo double `(x, y)`, la solución es multiplicar cada atributo `x` e `y` con la velocidad. Y luego sumar el resultado a la posición.
Otro conflicto que surgió es que en la clase world no sabíamos si declarar el valor marge en el atributo directamente o inicializar ese valor en el constructor `World`. Al final lo acabamos inicializando en el constructor World porque se veía más limpio y ordenado

5.3 La simulación de los agentes funciona correctamente, podrían añadirse más agentes al canvas y seguiría funcionando, al igual que modificar su velocidad de 1px/s.

Tuvimos cierta dificultad en la implementación del ejercicio opcional, el verificar colisión entre agentes con el método de la clase Agent `isColliding()` que verifica dicha colisión y en la clase World, `manageCollisions()` que gestiona la colisión para cada agente, otorgando un nuevo objetivo al agente si hay colisión. Esto produce que los agentes a veces respondan bien ante dicha colisión, cuando el camino del nuevo objetivo generado está libre y ambos agentes cambian de dirección, sin embargo si el objetivo generado sigue coincidiendo en la trayectoria con el otro agente se genera un conflicto.