

Lab 01: Implementing a Class

UPF-POO22-G202-21

Ariadna Prat / Chris Matienzo

3. The Agent and World Classes

3.1 The Agent Class

3.2 The World Class

5. Documentation

5.1 El programa consiste en simular el movimiento de un número de círculos (`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 posición 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 ()` // Método 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 width, 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 colisión entre los mismos 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.

Una dificultad en la implementación podría encontrarse en la clase `Agente`, donde el método de comparación entre agentes `isColliding`, por la sintaxis de como ingresar a los parametros de diferentes agentes para compararlos/operarlos entre sí.

```
public boolean isColliding(Agent agent) {
    Vec2D new_vector = new Vec2D(this.pos); // creamos un vector de clase Vec2D
    con los datos de posicionamiento
    new_vector.subtract(agent.pos); // Calculamos la diferencia de posiciones entre
    agentes con el método subtract
    double length = new_vector.length() + agent.pos.length(); // Calculamos la
    distancia entre dos agentes como la suma de ambas distancias
    double radius = this.radius + agent.radius; // Calculamos la suma del radio de
    ambos agentes
    if (length < radius) { // Si la distancia es menor que la suma del radio de ambos,
    entonces colisiona, de otro modo, no
        return true;
    } else {
        return false;
    }
}
```