Juan Pablo González Casado
Erasmus in University of Piraeus

Packet's World - Single reactive agent

Introduction

The problem is the simulation of an agent that is able to search for packages over land and take them to their destination once loaded.

For its development, it has a deterministic and static environment. This means that the limits of the environment are known and will not change after running the program.

The other important for the execution of such tasks point is the use of a reactive agent in charge of managing the possible targets as the knowledge gained so far.

Environment:

The setting is determined by a matrix of cells, in which each cell can take various types of roles (obstacle, package or agent). The default environment size is 50x50.

For the built environment, they have created different Java classes to facilitate handling and management.

Class Coordinates:

This class establishes the ability to create a coordinate element, thus containing the X coordinate and Y coordinate

Class Cell:

This class contains the information necessary to give useful values to each cell. The values used are:

- **Cell Value**: Indicating if empty, obstacle, package or agent.
- **Packet**: In case of a cell with value package, this variable will contain a package.
- **Potential**: Cell value used for decision-making potential.

Juan Pablo González Casado
Erasmus in University of Piraeus

Class Packet:

The packet class contains the necessary information about a package:

- **Color**: This attribute is only identification. Each color has an associated weight.
- **Weight**: The weight is determined by the color of the package. There are some constants that indicate that weight of each color.
- **Destination**: Indicates the destination coordinate which has to be carried package.
- **Moved**: Indicates whether the package has already been moved or not.

Class Constants:

The Constants class contains constants to configure the program:

- Match each color with its weight

```
//Colors constants
   public static final Packet white = new Packet("white",1);
   public static final Packet yellow = new Packet("yellow",5);
   public static final Packet green = new Packet("green",10);
   public static final Packet blue = new Packet("blue",20);
   public static final Packet red = new Packet("red",50);
   public static final Packet black = new Packet("black",100);
```

- World size:

```
//World constants
   public static final int worldsize = 50;
```

- Agent's contants:

```
   public static final int delta = 4;
   public static final int stamina = 1000;
   public static final int restTime = 10000;
```

- Cell's values:

```
//Cell constants
public static final int cell_empty = 0;
public static final int cell_filled = 1;
public static final int cell_obstacle = 2;
public static final int cell_agent = 3;
```

- Action id's:

```
//ActionType constants
public static final int actForward = 0;
public static final int actTurnRight = 1;
public static final int actTurnLeft = 2;
public static final int actPickup = 3;
public static final int actPutdown = 4;
public static final int actRest = 5;
```

- Directions:

```
//Direction
public static final int North = 0;
public static final int East = 1;
public static final int South = 2;
public static final int West = 3;
```

- Backtracking depth:

```
public static final int limitBacktracking = 45;
```

The setting alone is responsible for receiving the actions the agent decides to take, modifying data and the position of agent, targeting agent and packages on the map.

It is also responsible for providing information to the agent, as can be the field of possible vision for the agent.

This type of functions are performed basically in two calls:

This function is responsible for collecting a matrix size view of the agent, with a particular agent position on the map.

```
public Cell[][] getVision(Coordinates agent_coordinates)
```

The other function is responsible for managing actions indicating the agent modifying the environment according to the type of action.

```
public void readAction(int action) throws InterruptedException
```

Agent

This is the most important part of the program.

The agent is responsible for collecting the data that the environment provides you and thanks to that take the most optimal decisions possible.

For decision making they have been used two important methods:

Potential:

This method is used when the agent does not have any package within your line of sight and has to make a proper decision to explore the map.

It is based primarily on an integer value assigned to each cell, increasing each time the agent performs an action. Each time the agent passes through the cell, the value of this potential is reset to 0.

For decision-making by this method, it is only necessary to check the possibility of going to one of the four squares surrounding the agent, choosing the higher value box, as it will be the longest without being accessed state.

Backtracking:

This is the method that has been used to find the optimal path to a package.

This is a recursive function that generates all possible paths to the target and choose the least number of steps.

This program has been limited to 45 maximum steps, since the computation time needed to check all roads in a 50x50 matrix is very large.

Juan Pablo González Casado
Erasmus in University of Piraeus

The main function, responsible for decision-making is as follows:

```
public int Think()
```

This function handles the different options that can make the agent according to the state where you are:

If you are in the position of the destination of the package has loaded, you should let it and indicate that action is actPutdown

```
//PUTDOWN PACKET

If ( packet != null && next_coor_dest != null && position.getX()==next_coor_dest.getX() &&
position.getY() == next_coor_dest.getY() )

    {

       next = Constants.actPutdown;

       next_coor_dest = null;

       next_coor = null;

       packet = null;

    }
```

If the agent has a loaded package and want to take your destination, you must calculate the path through backtracking.

```
//Leading the packet

else if(packet != null)

    {

       Backtracking back_dest2 = new Backtracking(position,packet.getDestination(),memory);

       back_dest2.backtracking(Direction,new Coordinates(position.getX(),position.getY()),0);

       next_coor_dest = packet.getDestination();

       n = back_dest2.getNext();

       next = getNextMovement(n);

    }
```

If the agent does not have any loaded package and is over the package considered more optimal capture; actPickup.

```
//Above the packet for pick up

else if(packet == null && next_coor != null && position.getX()==next_coor.getX() &&
position.getY() == next_coor.getY())

    {

       packet = packetpickup;

       next = Constants.actPickup;

    }
```

Juan Pablo González Casado
Erasmus in University of Piraeus

If no packages in the line of sight of the agent, a target calculation is performed by the method of potential.

```
else if(visiblePackets.isEmpty())//      {
        next = ThinkByPotential();
    }
```

In case there are packages in the line of sight of the agent, it must perform a calculation by backtracking to see which is closer.

one counting on the package weight and destination calculation, so that in the case of a very heavy to a distant destination package, lose preference to a little heavy package with nearby destination even if you are further also realize that the heavy .

```
else//There are packets in the vision
    {
        for(Coordinates c : visiblePackets)
        {

            Backtracking back = new Backtracking(position,c,memory);
            back.backtracking(Direction,new Coordinates(position.getX(),position.getY()),0);
            Packet p = memory[c.getX()][c.getY()].getPacket();
            weight = p.getWeight();
            Backtracking back_dest = new Backtracking(c,p.getDestination(),memory);
            back_dest.backtracking(directionPath(position,c),new
Coordinates(c.getX(),c.getY()),0);
            int x=back.getCamino()+(back_dest.getCamino()*weight);
            if(back.getCamino()==0)
            {
                next = ThinkByPotential();
            }
            else if(x<tamano)//Path with less cost
            {
                n = back.getNext();
                tamano = x;
                next_coor = c;
                packetpickup = memory[c.getX()][c.getY()].getPacket();
                next = getNextMovement(n);
            }
        }
```

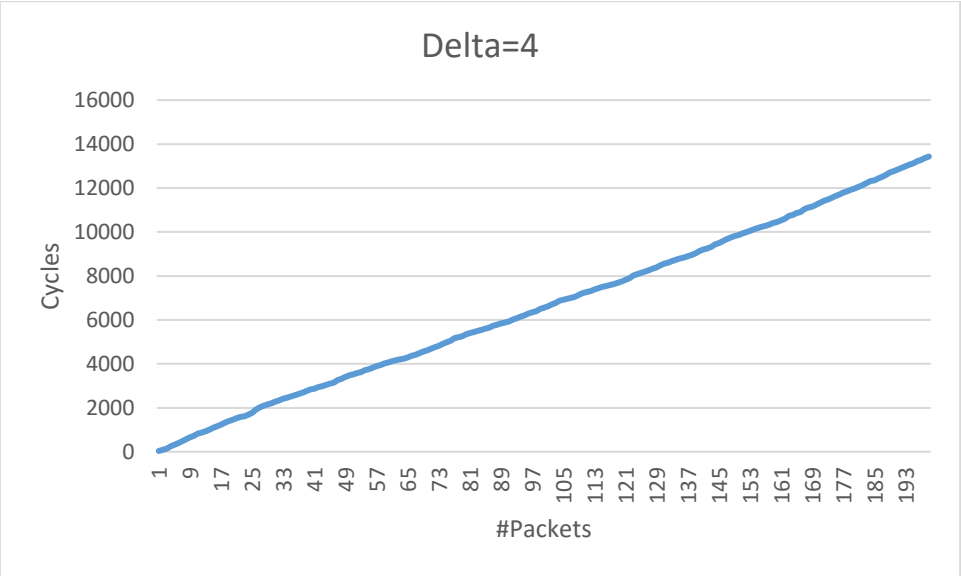
```
}
```

Finally, whenever an action is performed, estimina agent is removed. In case of a loaded packet, the corresponding weight of the package will be removed, however, only one will be deleted.

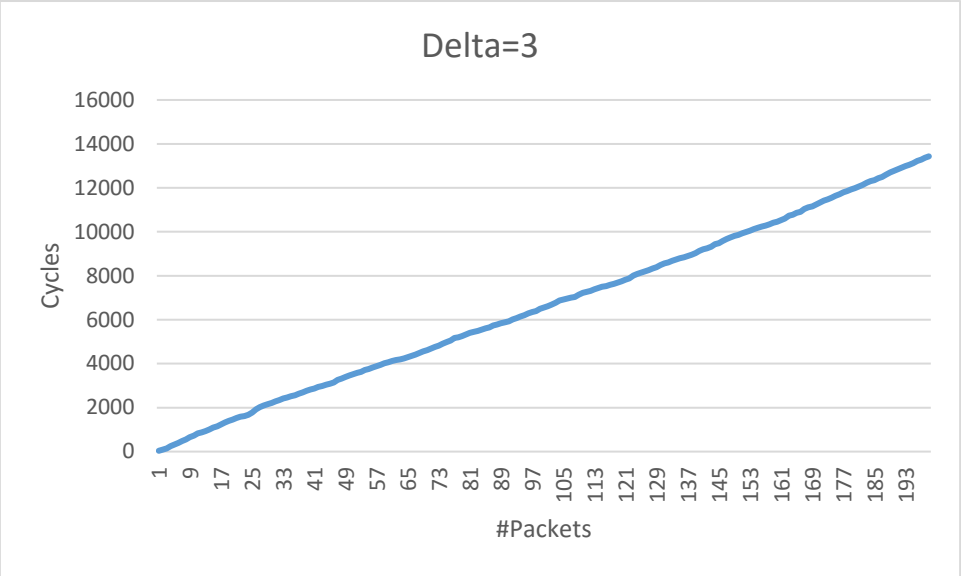
When the value of stamina is below 0, actRest is returned and the program is blocked for 10 seconds.

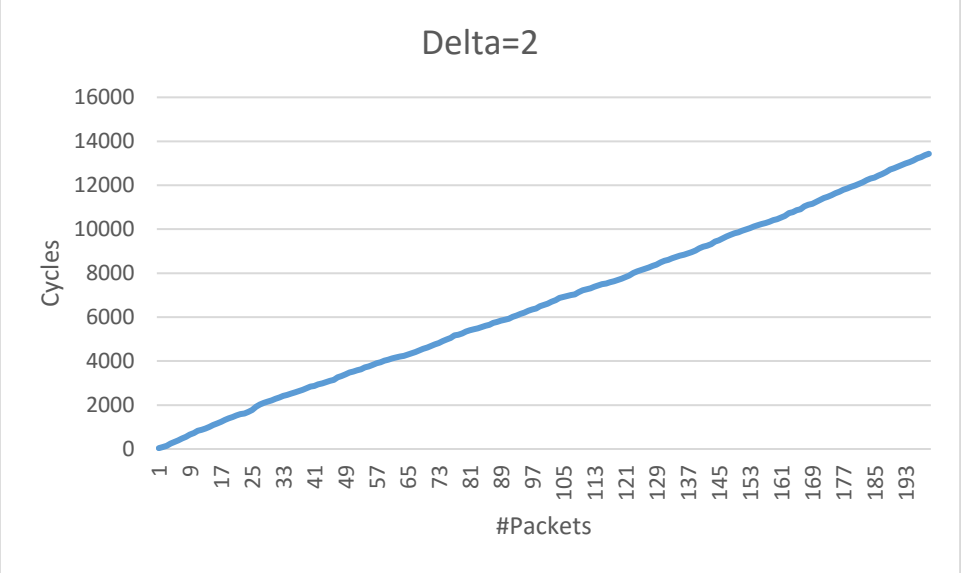
```
//Stamina
if(packet!=null)
   this.stamina-=packet.getWeight();
else
   this.stamina--;
System.out.println("Stamina="+this.stamina);
if(this.stamina<=0)
{
   next = Constants.actRest;
   this.stamina = Constants.stamina;
}
```

Charts



Juan Pablo González Casado
Erasmus in University of Piraeus





Juan Pablo González Casado
Erasmus in University of Piraeus