Intelligent Agents

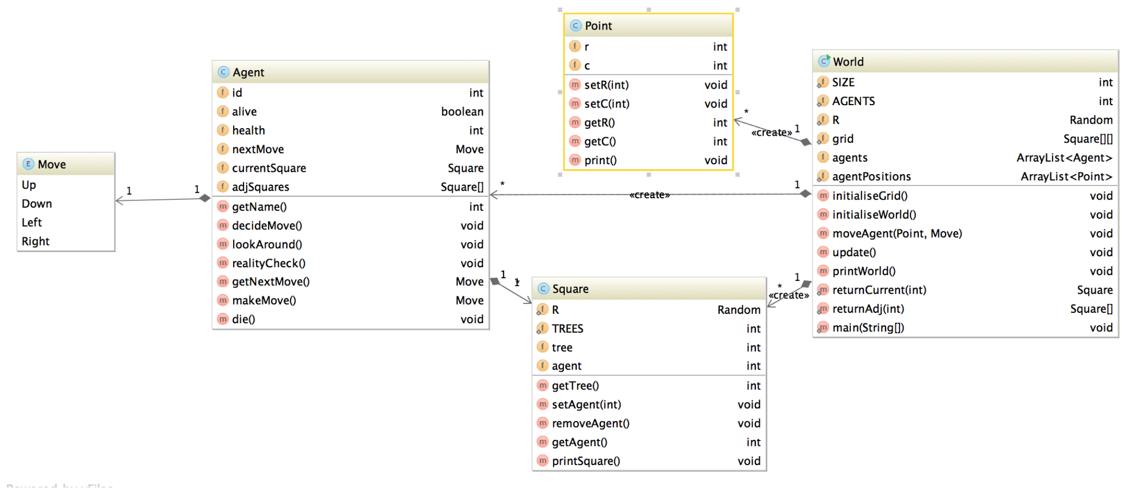
Antonis Pishias

Stage 1

Stage 1 requirements

- World : a grid of squares
- Square : contains a single tree (several kinds of tree exist)
- Robot (agent) : in a square / only one in each square at a time (otherwise crash)
- Time in ticks
- On each tick, each robot can do nothing, or perform a single action.
- A robot can move ahead, or rotate one quarter turn left or right.
- A robot can always see the tree in the square it is in, and they can look and see the tree in any adjacent square.
- Robots can recognise the different sorts of tree.

Solutions



Class diagram

As per the requirements the world and the agents are different classes and the agents never know the coordinates of their position since the squares themselves are just a class and the coordinates are held by a 2 dimensional array of Squares (grid). To make it easier to locate the agents in the world there is an agentPositions array list that holds the point where each agent is placed. This array list is continuously updated in case any agent move to a different place.

An enum class represents the possible moves an agent can make at any point. There is a small diversion from what the instructions asked for that I think is a trivial one. In my implementation the agents do not rotate left and right they can move UP, DOWN, LEFT and RIGHT at any position that this is possible. They can also see the squares, and what is in them, in the aforementioned directions but to do so they have to ask the World to return what Squares are around them (see returnAdj) the array of Squares returned is in a predefined order an any direction that does not have a square that the agent can move in is returned as null. To see the current square (see returnCurrent).

The world has a main function that initialises it and then while the user chooses to move on to the next tick it runs the update function that first asks the agents to make their mind, perform a reality check:

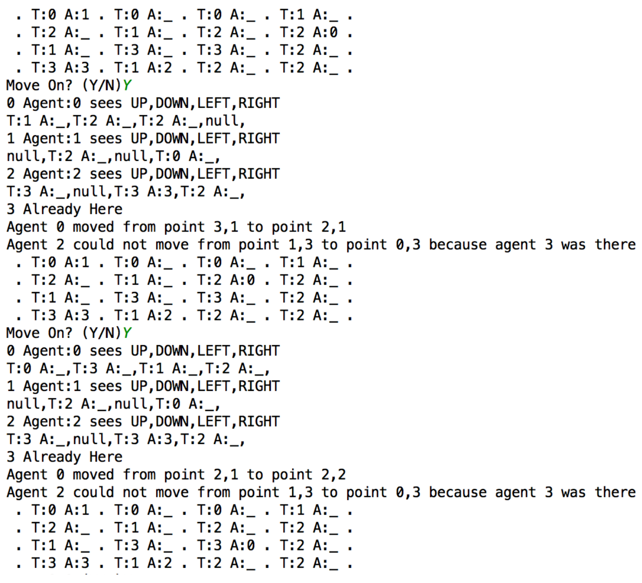
```
public void realityCheck() {
      currentSquare = World.returnCurrent(id);
      decideMove();
}
```

and then once all the agents decide on where they want to move they are asked to perform their action and the world decides if this is a legal action or not

For this stage the robots have a goal to move to a square with a tree of a certain kind (3) if they see a tree of kind 3 they decide that their next move will be that tree else they decide on a random move possible move. When time comes to move all agents try to move and any conflicts are made visible.

What happens if two robots try to move on to the same square at once?
The fastest robot makes it to the square (for now there is nothing that points out the speed of the robots so I assume that the first one in the array is the fastest one)

Example Run



```
 . T:0 A:1 . T:0 A:_ . T:0 A:_ . T:1 A:_ .
 . T:2 A:_ . T:1 A:_ . T:2 A:_ . T:2 A:0 .
 . T:1 A:_ . T:3 A:_ . T:3 A:_ . T:2 A:_ .
 . T:3 A:3 . T:1 A:2 . T:2 A:_ . T:2 A:_ .
Move On? (Y/N)Y
0 Agent:0 sees UP,DOWN,LEFT,RIGHT
T:1 A:_,T:2 A:_,T:2 A:_,null,
1 Agent:1 sees UP,DOWN,LEFT,RIGHT
null,T:2 A:_,null,T:0 A:_,
2 Agent:2 sees UP,DOWN,LEFT,RIGHT
T:3 A:_,null,T:3 A:3,T:2 A:_,
3 Already Here
Agent 0 moved from point 3,1 to point 2,1
Agent 2 could not move from point 1,3 to point 0,3 because agent 3 was there
 . T:0 A:1 . T:0 A:_ . T:0 A:_ . T:1 A:_ .
 . T:2 A:_ . T:1 A:_ . T:2 A:0 . T:2 A:_ .
 . T:1 A:_ . T:3 A:_ . T:3 A:_ . T:2 A:_ .
 . T:3 A:3 . T:1 A:2 . T:2 A:_ . T:2 A:_ .
Move On? (Y/N)Y
0 Agent:0 sees UP,DOWN,LEFT,RIGHT
T:0 A:_,T:3 A:_,T:1 A:_,T:2 A:_,
1 Agent:1 sees UP,DOWN,LEFT,RIGHT
null,T:2 A:_,null,T:0 A:_,
2 Agent:2 sees UP,DOWN,LEFT,RIGHT
T:3 A:_,null,T:3 A:3,T:2 A:_,
3 Already Here
Agent 0 moved from point 2,1 to point 2,2
Agent 2 could not move from point 1,3 to point 0,3 because agent 3 was there
 . T:0 A:1 . T:0 A:_ . T:0 A:_ . T:1 A:_ .
 . T:2 A:_ . T:1 A:_ . T:2 A:_ . T:2 A:_ .
 . T:1 A:_ . T:3 A:_ . T:3 A:0 . T:2 A:_ .
 . T:3 A:3 . T:1 A:2 . T:2 A:_ . T:2 A:_ .
```

Each Square is represented by the value of its tree T and the agent A. In this specific example we can see Agent 0 moving randomly around until it finds a square with a tree of value 3 where its journey ends.

To make what happens more transparent each agent outputs what it sees and the world logs all the actions and if they were success full.

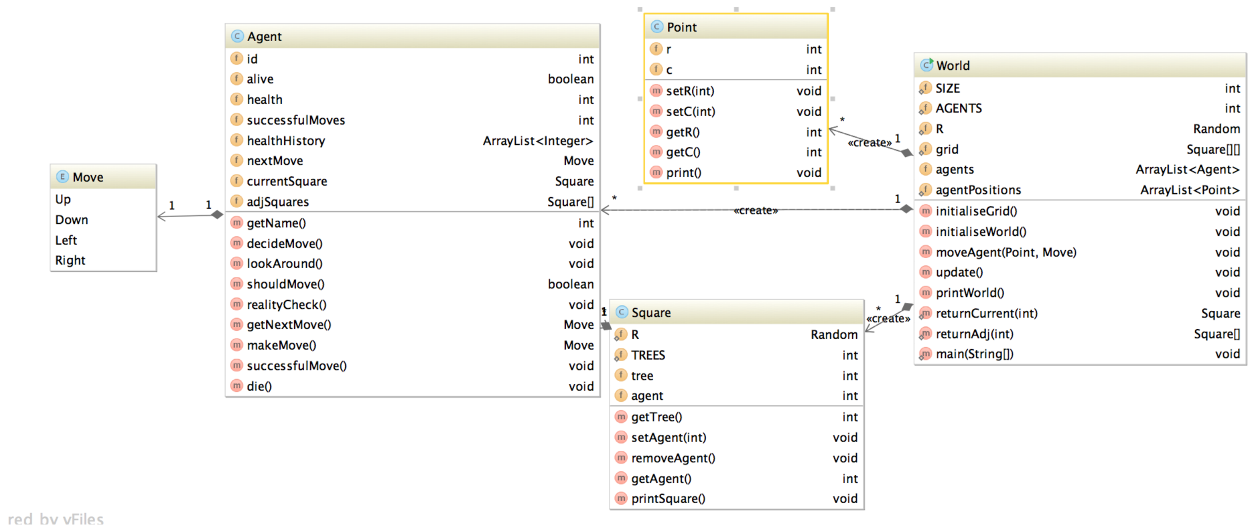
Another thing that is visible here is how Agent 2 always tries to move towards the left and always fails. This could be avoided by making it possible to remember that that move did not work and so try something different. This would require some memory location (array of all moves within the agent)

Stage 2

Stage 2 requirements

- Robots have a health level (goes down one unit per tick)
- If the health level reaches zero, the robot dies.
- A robot can eat a fruit from the tree of the square they are in.
- The level goes up, or down, or is unaffected, when they eat the fruit of certain kinds of tree - but they do not initially know what the effect of each sort of fruit on them is.
- After eating each sort of fruit, the robot can notice the change in its health level.
- At this stage, the robots' goal is to live as long as possible.
- At each tick, a robot has to work out what to do. How does it do that at this stage?

Solutions



Class diagram

The new field in the Agent Class are alive, successfulMoves and healthHistory accompanied by some new methods shouldMove, which is now part of decideMove and successfulMove that increments the number of successfulMoves once one is made.

Each square has a tree. Each tree has a value that is deducted from the players health when in that square. The robot can judge if the deduction is more acceptable from any other previous deduction and the deduction of the tick. If acceptable the robot will choose to stay at the square if not it will choose to move somewhere different or move back to a square that costs less.

The decision if the value deducted is acceptable is decided within the new shouldMove function where the average deduction is calculated and if the last deduction is less or equal to that value then i decides that not moving is a better option than moving. For this to work though the agent should explore a bit first because if it does not the deductions will always be the same and the best action would be to stay in the starting square. So the algorithm forces the agent to move in the beginning and then starts to work as described above.

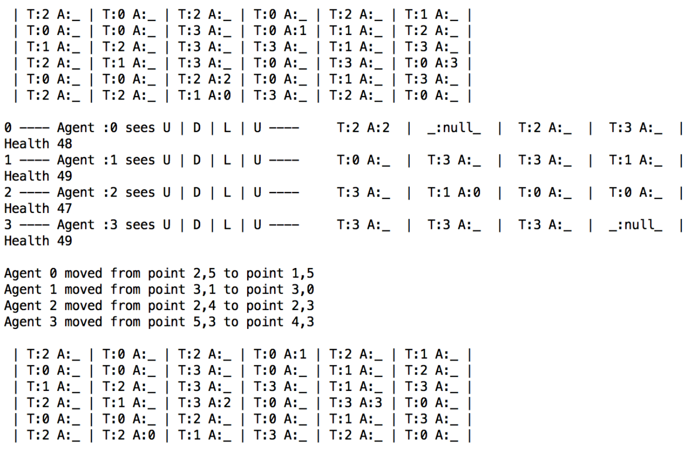
Essentially the behaviour here is explore to get a feel of what is happening in the world and then if the current square offers less damage than the average stay there. Another tactic would be to try and find again the square where the minimum damage was caused.

```
boolean shouldMove() {
    if (healthHistory.size()>1 && successfulMoves>0) {
        //if it explored compare
        int sum =0;
        for (Integer aHealthHistory : healthHistory) {
            sum += aHealthHistory;
        }
        int average = sum/healthHistory.size();
        if (average>=healthHistory.get(healthHistory.size()-1)) return false;
        else return true;

    } else {
        //if it did not move yet it should (explore)
        return true;
    }
}
```

```
public void realityCheck() {
    if (alive) {
        currentSquare = World.returnCurrent(id);
        health--;
        if (health<0) die();
        decideMove();
    }
    else {
        System.out.println("Agent "+id+" is dead");
    }
}
```

Example Run



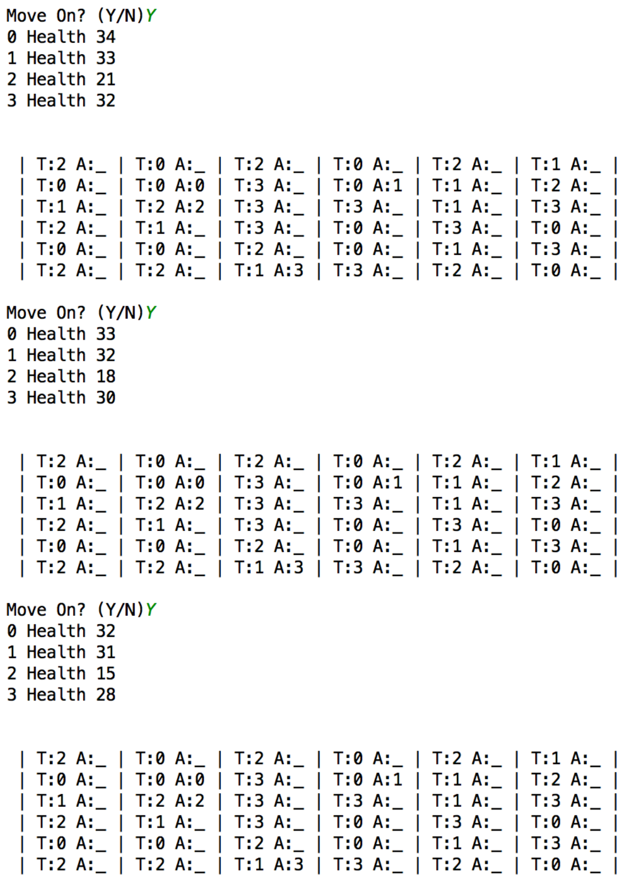
```
 | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:1 A:_ |
 | T:0 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:1 | T:1 A:_ | T:2 A:_ |
 | T:1 A:_ | T:2 A:_ | T:3 A:_ | T:3 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:1 A:_ | T:3 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:3 |
 | T:0 A:_ | T:0 A:_ | T:2 A:2 | T:0 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:2 A:_ | T:1 A:0 | T:3 A:_ | T:2 A:_ | T:0 A:_ |

0 ---- Agent :0 sees U | D | L | U ----     T:2 A:2  |  _:null_  |  T:2 A:_  |  T:3 A:_  |
Health 48
1 ---- Agent :1 sees U | D | L | U ----     T:0 A:_  |  T:3 A:_  |  T:3 A:_  |  T:1 A:_  |
Health 49
2 ---- Agent :2 sees U | D | L | U ----     T:3 A:_  |  T:1 A:0  |  T:0 A:_  |  T:0 A:_  |
Health 47
3 ---- Agent :3 sees U | D | L | U ----     T:3 A:_  |  T:3 A:_  |  T:3 A:_  |  _:null_  |
Health 49

Agent 0 moved from point 2,5 to point 1,5
Agent 1 moved from point 3,1 to point 3,0
Agent 2 moved from point 2,4 to point 2,3
Agent 3 moved from point 5,3 to point 4,3

 | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:1 | T:2 A:_ | T:1 A:_ |
 | T:0 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:_ | T:1 A:_ | T:2 A:_ |
 | T:1 A:_ | T:2 A:_ | T:3 A:_ | T:3 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:1 A:_ | T:3 A:2 | T:0 A:_ | T:3 A:3 | T:0 A:_ |
 | T:0 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:2 A:0 | T:1 A:_ | T:3 A:_ | T:2 A:_ | T:0 A:_ |
```

Moving Randomly at the beginning (above)
Settling where there is less damage than average (right)



```
Move On? (Y/N)Y
0 Health 34
1 Health 33
2 Health 21
3 Health 32

 | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:1 A:_ |
 | T:0 A:_ | T:0 A:0 | T:3 A:_ | T:0 A:1 | T:1 A:_ | T:2 A:_ |
 | T:1 A:_ | T:2 A:2 | T:3 A:_ | T:3 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:1 A:_ | T:3 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:_ |
 | T:0 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:2 A:_ | T:1 A:3 | T:3 A:_ | T:2 A:_ | T:0 A:_ |

Move On? (Y/N)Y
0 Health 33
1 Health 32
2 Health 18
3 Health 30

 | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:1 A:_ |
 | T:0 A:_ | T:0 A:0 | T:3 A:_ | T:0 A:1 | T:1 A:_ | T:2 A:_ |
 | T:1 A:_ | T:2 A:2 | T:3 A:_ | T:3 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:1 A:_ | T:3 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:_ |
 | T:0 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:2 A:_ | T:1 A:3 | T:3 A:_ | T:2 A:_ | T:0 A:_ |

Move On? (Y/N)Y
0 Health 32
1 Health 31
2 Health 15
3 Health 28

 | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:2 A:_ | T:1 A:_ |
 | T:0 A:_ | T:0 A:0 | T:3 A:_ | T:0 A:1 | T:1 A:_ | T:2 A:_ |
 | T:1 A:_ | T:2 A:2 | T:3 A:_ | T:3 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:1 A:_ | T:3 A:_ | T:0 A:_ | T:3 A:_ | T:0 A:_ |
 | T:0 A:_ | T:0 A:_ | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:3 A:_ |
 | T:2 A:_ | T:2 A:_ | T:1 A:3 | T:3 A:_ | T:2 A:_ | T:0 A:_ |
```

Stage 3

Stage 3 requirements

- Robots map as much of the world as possible.
- Know what sort of tree is on which square. (memory)
- There is no time signal
- There is no GPS

What do they remember?
How do they remember it?

In this stage the memory will be a list of all the successfulMoves taken and the tree found after that move. This could be implemented as a list of objects. A new kind of object should be created that holds the move made (and maybe the reverse needed to be made to go back) as well as the tree at the location the agent just left, also the tree that will be found when the reverse move will be executed.

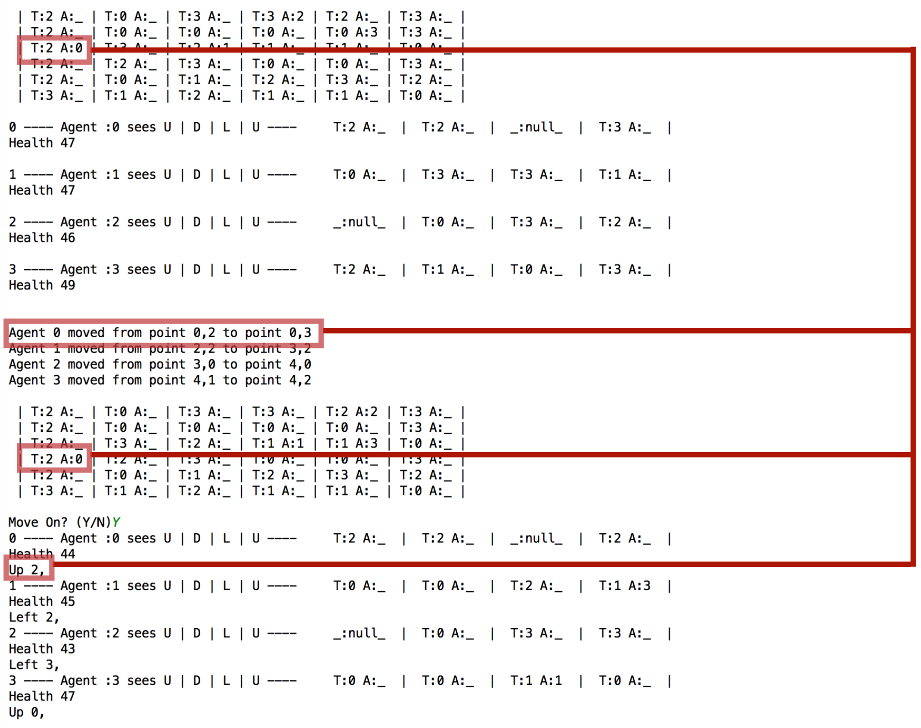
```
class Memory {
   Move moveMade;
   Move reverseMove;
   int treeFound;

   public Memory(Move m, int t) {
      moveMade = m;
      reverseMove = returnReverse(m);
      treeFound = t;
   }
}
```

When the agent decides the move that it should make it creates a potential memory which is added to the actual memory stack, in this case called "past", if and only if the move is successfully completed. This way the only memories created are moves that have taken place.

```
public Move makeMove() {
   Move currentMove = nextMove;
   currentMemory = new Memory(currentMove, currentSquare.getTree());
   nextMove = null;
   return currentMove;
}

public void successfulMove() {
  past.add(currentMemory);
   successfulMoves++;
}
```



```
 | T:2 A:_ | T:0 A:_ | T:3 A:_ | T:3 A:2 | T:2 A:_ | T:3 A:_ |
 | T:2 A:_ | T:0 A:_ | T:0 A:_ | T:0 A:_ | T:0 A:3 | T:3 A:_ |
 | T:2 A:0 | T:3 A:_ | T:2 A:1 | T:1 A:_ | T:1 A:_ | T:0 A:_ |
 | T:2 A:_ | T:2 A:_ | T:3 A:_ | T:0 A:_ | T:0 A:_ | T:3 A:_ |
 | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:2 A:_ | T:3 A:_ | T:2 A:_ |
 | T:3 A:_ | T:1 A:_ | T:2 A:_ | T:1 A:_ | T:1 A:_ | T:0 A:_ |

0 ---- Agent :0 sees U | D | L | U ----     T:2 A:_  |  T:2 A:_  |  _:null_  |  T:3 A:_  |
Health 47

1 ---- Agent :1 sees U | D | L | U ----     T:0 A:_  |  T:3 A:_  |  T:3 A:_  |  T:1 A:_  |
Health 47

2 ---- Agent :2 sees U | D | L | U ----     _:null_  |  T:0 A:_  |  T:3 A:_  |  T:2 A:_  |
Health 46

3 ---- Agent :3 sees U | D | L | U ----     T:2 A:_  |  T:1 A:_  |  T:0 A:_  |  T:3 A:_  |
Health 49


Agent 0 moved from point 0,2 to point 0,3
Agent 1 moved from point 2,2 to point 3,2
Agent 2 moved from point 3,0 to point 4,0
Agent 3 moved from point 4,1 to point 4,2

 | T:2 A:_ | T:0 A:_ | T:3 A:_ | T:3 A:_ | T:2 A:2 | T:3 A:_ |
 | T:2 A:_ | T:0 A:_ | T:0 A:_ | T:0 A:_ | T:0 A:_ | T:3 A:_ |
 | T:2 A:_ | T:3 A:_ | T:2 A:_ | T:1 A:1 | T:1 A:3 | T:0 A:_ |
 | T:2 A:0 | T:2 A:_ | T:3 A:_ | T:0 A:_ | T:0 A:_ | T:3 A:_ |
 | T:2 A:_ | T:0 A:_ | T:1 A:_ | T:2 A:_ | T:3 A:_ | T:2 A:_ |
 | T:3 A:_ | T:1 A:_ | T:2 A:_ | T:1 A:_ | T:1 A:_ | T:0 A:_ |

Move On? (Y/N)Y
0 ---- Agent :0 sees U | D | L | U ----     T:2 A:_  |  T:2 A:_  |  _:null_  |  T:2 A:_  |
Health 44
Up 2,
1 ---- Agent :1 sees U | D | L | U ----     T:0 A:_  |  T:0 A:_  |  T:2 A:_  |  T:1 A:3  |
Health 45
Left 2,
2 ---- Agent :2 sees U | D | L | U ----     _:null_  |  T:0 A:_  |  T:3 A:_  |  T:3 A:_  |
Health 43
Left 3,
3 ---- Agent :3 sees U | D | L | U ----     T:0 A:_  |  T:0 A:_  |  T:1 A:1  |  T:0 A:_  |
Health 47
Up 0,
```

Agent 0 remembers that if it moves UP it will be at a square with a tree of kind 2