ESE 124 Programming Fundamentals
Prof. Alex Doboli

Course Project
Spring 2021

Michael: The Greedy Ant

Episode 2

Overview of Episode 1: In this episode, Ant Michael had to effectively find its way out of a maze. Of course, Michael did not know the structure of the maze. Moreover, it had to find a way out in the smallest number of steps and with the smallest amount of spent energy. Otherwise, it might run out of energy and perish.

Description of Episode 2:

Implement a C program with the following description.

1. **The description of VA (Virtual Ant) – a greedy ant**

Michael, our greedy ant, is actually a virtual ant (called for brevity from now on just Michael).

- Michael has its own memory implemented as a stack of size *MAX_SIZE* and with the operators (functions) like those discussed in class: pop, push, peek, clear, empty and full. Each stack element can store the current physical position of the ant inside the maze. The physical position is described as the pair **x-y**. The positions are indicated by 2 integer numbers, **x**, **y**, but the accurate position is not displayed to Michael. So, Michael does not know the map of the place it must navigate and it does not know its current position or any of its previous positions.
- Michael knows to perform the following actions:
    1. MARK – the ant marks its current position using a chemical called pheromone.
    2. MOVE_F – moves the VA from the current position one position forward. If Michael locates in (x, y), it will move to (x + 1, y).
    3. MOVE_B – moves the virtual ant from the current position one position backward. If Michael locates in (x, y), it will move to (x - 1, y).
    4. MOVE_L – moves the virtual ant from the current position one position left. If Michael locates in (x, y), it will move to (x, y - 1).

5. MOVE_R – moves the virtual ant from the current position one position right. If Michael locates in (x, y), it will move to (x, y + 1).

6. CWL – Michael checks if the next locations (until meeting a wall) to the left are pheromone free. If the locations are free then Michael feels an itch. Otherwise, if no location is free (e.g., because there is a pheromone mark or a wall on the left of Michael), then Michael does not feel the itch.

7. CWR – Michael checks if the next locations (until meeting a wall) to the right are pheromone free. If the locations are free then Michael feels another kind of itch. Otherwise, if no location is free (e.g., because there is a wall or a pheromone mark on the right of Michael), then Michael does not feel the itch.

8. CWF – Michael checks if the next locations (until meeting a wall) in front are pheromone free. If the locations are free then Michael feels a third kind of itch. Otherwise, if no location is free (e.g., because there is a wall or a pheromone mark in front of Michael), then Michael does not feel the itch.

9. CWB – Michael checks if the next locations (until meeting a wall) backwards are pheromone free. If the locations are free then Michael feels a fourth kind of itch. Otherwise, if no location is free (e.g., because there is a wall or a pheromone mark behind of Michael), then Michael does not feel the itch.

10. PUSH – pushes the planar coordinates x and y of Michael's current position into Michael's stack for the memory. This way Michael memorizes the current position.

11. POP - pops the planar coordinates x and y from the top of the Michael's stack for the memory. This way Michael remembers the position, but then it immediately forgets the position too.

12. PEEK - peeks the planar coordinates x and y from the top of the Michael's stack for the memory. This way Michael remembers the position, but does not forget the position.

13. CLEAR – Michael clears its stack. This way it has a total amnesia and forgets all its previous memories.

14. BJPI (Bold jump for itching) – jump x position along the direction for which Michael felt an itch (left, right, forward, backward). For example, after performing CWR, Michael felt an itch because the direction to the right of the current position was free. Then it decided to act using BJPI, meaning that it jumped x positions to the right, as the itching was felt after Michael checked the locations to the right of its current position. The number of positions over which it jumped is found by after using the corresponding CWL, CWR, CWF, CWB action. For example, if the next 4 locations to the left of the current position are free, then x is 4 after executing CWL. If x is zero after executing action CWL, but the ant still executes action BJPI, then Michael stays in its current position. Every BJPI stops the corresponding itching of the ant, e.g., the itching type that triggered the jump.

15. CJPI (cautious jump for itching) – jump one position along the direction for which Michael felt an itch (left, right, forward, backward). For example, after performing CWR, Michael felt an itch because the direction to the right of the current position was free. Then it decided to act using CJPI, meaning that it jumped one positions to the right, as the itching was felt after Michael checked the locations to the right of its current position. Every CJPI

2. Program description

- A number of pots of gold are placed at different positions inside a maze.
- Michael must traverse a labyrinth and collect as many pots of gold as possible before running out of energy. Therefore, it must use an intelligent strategy. The labyrinth has one entry and Michael enters the labyrinth using the entry.
- Michael's intelligence is described as a sequence of actions that Michael executes to find its path to the pots of gold. You (meaning you, the student taking ESE 124) describe the sequence of actions in an input file, which your C program reads before requiring Michael to execute the actions.
- The labyrinth to be traversed is read from a file.
- The sequence of actions performed by Michael is written into an output file together with the spent energy to navigate the labyrinth.

3. The greedy ant might consumes energy during its search

- The amount of energy is constant *MAX_ENERGY*. Every time Michael performs one of the actions 1-5 it consumes 3 units of energy. Every time Michael performs one of the actions 6-9 it consumes 1 unit of energy. Every time Michael performs one of the actions 10-11 it consumes 4 units of energy. Every time VA performs one of the actions 12-13 it consumes 2 units of energy. Every time VA performs one of the actions 14 it consumes 5 units of energy. Action 15 consumes 3 units of energy. VA stops if its energy reaches zero.

4. Other program requirements

- Your program must create an Abstract Data Type (ADT) VA that implements all the actions of the ant. Your program must also create and Abstract Data Type (ADT) for the ant's stack.

5. Experiments

- Execute the program for different values of constants *MAX_SIZE* and *MAX_ENERGY*. Also, consider different values for *n* and *t* in the actions RP n t. Record for each case, the number of found pots of gold, the total number of actions, and the used energy (as recorded in the output file).