Advanced Topics in Programming

Koren Ben Ezra– Arbel Klein

Main

The main is responsible for running the simulator using multi-threading. It loads the algorithm's shared libraries and run the simulator on each algorithm-house pair that exists. Each algorithm-house pair get a score, and at the end the main write summary of the scores into a csv file.

The main can get 5 command line arguments:

- -house_path – The directory that contains the .house files that represents houses need to be cleaned. If not used, then the main search for .house files in current directory.
- -algo_path – The directory that contains the .so file that are the shared libraries of the algorithms. If not used, then the main search for .so files in current directory.
- -num_thread – The number of threads that the main can use at most. If not used, the number of threads is set to 10.
- -summary_only – Output files won't be created, only the csv file.
- -log – In order to create a log file for each algorithm-house pair.

The flow:

1. Load the algorithm's shared libraries
2. Load the houses
3. For each algorithm-house pair:
    a. If number of active threads is lower than the number of allowed threads:
        i. Create a thread
        ii. Run the simulator with the algorithm-house pair on the thread
        iii. Insert the thread into a queue
    b. Wait until some thread will finish
4. Wait for all threads to finish
5. Close all open algorithms
6. Write to summary.csv file

Error handling:

If any error occurred during the run of the main, an error file will be created (in errors directory in current working directory), and the error will be written in it.

Algorithm A – DFS movement

Main flow:

1. Explore all reachable spots through DFS algorithm while mapping the house.
2. Returning to the docking station through the shortest path found with BFS algorithm.
3. Deciding to finish at the docking station iff all reachable spots at range *maxBattery/2* are clean.

Algorithm B – Spiral movement

Main flow:

1. Explore all reachable spots in spiral movement
    a. Move right if can, else
    b. Move straight, else
    c. Move to the '*closest*' not cleaned or known unvisited spot
        i. With BFS to look for designated spots
        ii. Calculate two shortest paths with BFS current→spot, spot→docking.
2. Returning to the docking station through the shortest path found with BFS algorithm.
3. Deciding to finish at the docking station iff all reachable spots at range *maxBattery/2* are clean.

Denote *'closest'* as the minimum number of steps the robot need to take to go to the designated spot and back to docking.

Simulation

The simulation runs using input and output files only, it simulates the *steps* in the output file on the *house structure* in the input file while keeping track on the battery, dirt level on each location, robot's location at each iteration etc. There are important variables presented in the simulation (*step, battery, dirt left)* and the status is presented at the end.

The house structure's parts are instances of *Part* such that there are 5 derived classes (*PTWall, PTEdgeWall, PTRobot, PTBlank, PTDocking)* and each part is an instance of one of them. To *draw()* each part differently we need that kind of structure.

- The simulation **supports large houses**, meaning that given an input larger than the window, the frame will move with the robot at need.
- Each dirt level (0-9) is represented with at most 3 small dirt squares with randomized initialized location in the tile.
  Number of small dirt squares is calculated by:

$$f: [0,9] \longmapsto [0,3], \qquad f[n] = \left\lceil \frac{n}{3} \right\rceil$$

How to use

Run with:

```
python3 ./Simulation.py myHouse.house myOutput.txt
```

Pygame is required, installation via:

```
pip install pygame
```