Amazing Project Design Specifications

Siyuan Feng
Chris Leech
Jordan Hall
5/19/14
COSC50 Software Design and Implementation / Charles Palmer / 14Spring

Input:

-h hostname
-d difficulty
-n nAvatars

Example: startup -h pierce.cs.dartmouth.edu -d 3 -n 5

hostname (pierce.cs.dartmouth.edu):

This is the location of the server that will be handling the maze.

difficulty (3):

This is the difficulty of the maze, must be a value from 1 to 9. The maximum difficulty level is defined in a constant, in this case 9. Mazes 0-4 are fixed while mazes 5-9 are different every time they are generated.

nAvatars (5):

This is the number of avatars that will be put into the maze. The maximum number of avatars is defined in a constant, in this case 10.

Output:

The program will write to a "picture" file a series of ASCII depictions of the graph, with the location of the avatars and the walls of the maze. The program will also create a log of the runs, and upon solving the maze, will write the "maze completed" hash to that log.

The first line of the log file will have the MazePort, AvatarId, and the date and time. When the maze is solved, a special AM_MAZE_SOLVED message will be created and printed to the log.

Data Flow:

Startup script:
- Parse command line arguments
- Pass to server and avatars
- Create processes for each avatar

Avatars:

Both the master and the slave will move using the left hand algorithm. Once a slave finds the bread crumb trail, then instead of moving using that algorithm, it will just follow the trail. Once a master reaches the maximum number of moves its allocated, it will stop moving as well.

Bread Crumb Algorithm for the Master:

The master will always be avatar 0. The master will leave a path of "bread crumbs" for the avatar. These bread crumbs will be stored on a linked list that will be shared memory throughout all the avatar processes. Whenever avatar 0 successfully moves, it will store it's position on this linked list. Therefore each linked list will have a next pointer to the next position that it moved to from this current position. The concept is that whenever the position of one of the slaves is equivalent to the position of any position on the list of breadcrumbs, then instead of moving according to the left hand algorithm described below, it will follow the next pointer in that bread crumb list.

Master Max Number of Moves:

The master will move a set number of steps. This number will be determined by the dimensions of the maze, given when we receive the INIT_OK message, divided by a constant MAX_MASTERS_MOVES. We limit the number of moves the master can make to make sure that master sets a long enough path of "bread crumbs" so that it can be easily found, but also to make sure that the master does not keep moving away from the slaves after the slaves find a breadcrumb.

Left Hand Algorithm for the Slave and Master - "Keep my left hand on the maze and walk":

We first have an algorithm to determine the direction that the avatar is facing. We store the previous position of the avatar and the current position of the avatar. We compare the x, and y values of those two positions.

If the x coordinates are the same then we know the avatar moved along the y axis, and if the y coordinates are the same, then we know the avatar moved along the x axis. So given that we moved along one axis, then we compare the coordinates of the other axis.

If y of the current position is less than y of the previous position, then the avatar is going or facing south. If y of the current position is greater than y of the previous position, then the avatar is going or facing north. If x of the current position is less than x of the previous position, then the avatar is going or facing west. If y of the current position is greater than y of the previous position, then the avatar is going or facing east.

Given the direction, we always want to move in order of greatest to least priority, west, north, east, and then south. This is why it's called the left hand algorithm because we always want to move west or left when possible.

Data Structures:

A linked list that each slave holds that contains the list of past locations of the lead avatar - "crumbs" list.

Variables:

Slave:
Previous Position: XYPos prev
Current Position: XYPos cur
Direction: struct dir - direction the slave is facing for the left hand algorithm

Master:
Num_steps - Number of steps it's taken, to be incremented after very successful move
Direction: struct dir - direction the master is facing for the left hand algorithm
Current Position: XYPos cur

Pseudocode for Avatars:

```
// check args

// send initial OK message

// get size of maze from INIT_OK and use that to calculate max number of moves for the master

// If I am slave:

        // listen for next move and receive message

        // if we have won:

                // free memory and exit

        // if it is my move:

                // draw maze

                // record position of lead avatar (create breadcrumb).

                // if I haven't moved since last time:

                // check if I am on the lead avatar's breadcrumbs

                        // if so:

                                // get next move along breadcrumbs
                                // move next move
```

```
        // otherwise, get next move according to regular algorithm
        // use the "left hand" algorithm.
```

If I am master:

```
    // listen for next move and receive message

    // draw maze

    // if we have won:

        // write message to file
        // free memory and exit

    // if it is my move:

        // if I am out of "crumbs":
            // move null

        // move along set lead heuristic
```

Functions and Files

Files: avatar.c, master.c, startup.c

Util library: list.c, list.h