Design Specification

The basic flow of the program is as follows:

1. Check arguments
2. Connect to the server with the initial message
3. Get a unique key (long) by reconstructing the full path of the currently running executable and using ftok on it
4. Fork the program into all the avatars (add fork for a log function with a named pipe?)
5. In each child:
6. Connect and attach to shared memory using the key calculated before forking
7. If we are avatar 0, open the log file and initialize the global worldmap
8. Connect to the server and send an AM_AVATAR_READY
9. For each step, execute our algorithm.
10. When we all are together, free the memory and return to main function
11. Exit successfully
12. In the main thread:
13. wait for all the children threads to return.

Algorithms

1. Naive right hand wall following algorithm. Fills in the dead end as well.
   (This method is very inefficient)
2. Implements right hand wall following algorithm. Fills in the dead end. Avatars are more ‘adventurous’, biased towards exploring the unexplored regions of the maze. (An improvement on the previous algorithm, more efficient)
3. Implements right hand wall following algorithm. Fills in the dead end. Avatars are more ‘adventurous’, or have bias towards exploring the unexplored regions of the maze. There is also a flood filling method that determines whether all the avatars are connected or not. When the avatars are determined to be all connected to each other, the avatars will stop exploring the unvisited regions of the maze but it will continue with dead end filling until the Avatars converge. (An improvement on the previous two algorithms, this algorithm is our most efficient algorithm so far. It is significantly more efficient especially on the larger mazes)

Implementation Specification

Our maze solving program has several important components, including:

- startup.c (the ‘main’ function)
- agent.c/.h (called by the startup function in each child process)
- amazing.h (defines mesage type structure)
- worldmap.c/.h (maze to be solved is kept track of locally by this)
- mazealgorithm.c/.h (called by agent.c, 3 different algorithms present)

- queue.c/.h (called by algorithm 3 in mazealgorithm.c to flood fill for algorithm 3)
- log.c/.h (functions for logging various levels 0 to 6)
- network.c/.h (handles sockets, etc…)
- common.h (defines succcess, failure)

Major Data structures used:

Worldmap

- Defined in worldmap.h
- Contains data about the x and y length of the maze, and an array of Square structure. This array is one dimensional (i = y*xlen+x).

Square

- Defined in worldmap.h
- Contains information about each individual square block composing the maze. Holds information such as where the walls of the maze are and whether an avatar is able to access it.

Queue

- Defined in queue.h
- Contains address of a head Node and a tail Node
- Queue is essentially a doubly linked list, used for a breadth-first algorithm that marks any squares connected to an agent without having to recursively call a function, which would otherwise cause stack overflow on anything else than a small maze.

Node

- Defined in queue.h
- Composes the queue, which is a doubly linked list
- Holds addresses of the previous Node and next Node. Each Node holds XYPos structure

XYPos

- Defined in amazing.h
- Holds the x and y coordinate of an agent

Error Conditions Detected and Reported

- The program will run check on the input parameters and flags and report errors.

- If there missing parameter inputs or invalid parameter inputs the program will warn the user.
- The following format should be used to call startup:

```
Usage: startup -n nAvatars -d Difficulty -h Hostname [-a algorithm] [-l logLevel]

nAvatars
This is the number of avatars to be put in the maze. 1 to 10 are accepted.
Difficulty
This is the difficulty level of the maze to be solved. 0 to 9 are accepted.
Hostname
This is the address of the host server. pierce.cs.dartmouth.edu is used for this assignment.
[algorithm]
This is an optional flag used to specify the algorithm used to congregate the avatars together. 0 to 2 are a
[logLevel]    This is an optional flag used to specify the level of logging output. 0 to 6 are accepted with
0 being the least verbose and 6 being the most verbose. Level 3 is set as the default.
```

Tests and Expected Results

Here are a few sample VALID inputs and outputs:

```
Input:
startup -n2 -d2 -a2 -hpierce.cs.dartmouth.edu
Output:
iteration 20
+-+-+-+-+-+-+-+-+-+-+-+-+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . . . . . . . . . . |
+.+.+.+.+.+.+.+.+.+.+.+.+
| . . .   . . .X. . . . |
+.+.+.+ + +.+.+ +.+.+.+.+
| . . . |1. .X X| . . . |
+.+.+.+ + +.+ +-+.+.+.+.+
| . . . |  2 X|X| . . . |
+.+.+.+ +-+-+-+ +.+.+.+.+
| . . .  X X X X| . . . |
+-+-+-+-+-+-+-+-+-+-+-+-+

Moves to solve: 39
Size = 12x10
Algorithm = 2
# of Avatars = 2
Log process exited
main process exited
```

```
Input:
startup -h pierce.cs.dartmouth.edu -d1 -a2 -n2
Output:
iteration 25
+-+-+-+-+-+-+-+
|X|X X X| . |
+ + +.+-+.+.+.+
|X|X. . . . |
+ + +.+.+.+.+.+
|X X. . . . |
+ +.+.+.+.+.+.+
|X| . . . . |
+ +.+.+.+.+.+.+
|X| . . . . |
+ +-+-+.+.+.+.+
|2|X X| . . |
+ + +.+.+.+.+.+
|  1. . . . |
+.+.+.+.+.+.+.+
| . . . . . |
+.+.+.+.+.+.+.+
| . . . . . |
+-+-+-+-+-+-+-+

Moves to solve: 49
Size = 6x9
Algorithm = 2
# of Avatars = 2
Log process exited
main process exited
```

Here are a few sample INVALID inputs and outputs:

```
Input:
startup -h pierce.cs.dartmouth.edu -d9

Output:
Usage:
startup -n nAvatars -d Difficulty -h Hostname [-a algorithm] [-l logLevel]

Input:
startup -h pierce.cs.dartmouth.edu -d2 -n2 -a9

Output:
startup: invalid algorithm: (0 <= algorithm <= 2)
```

Notes on Memory Leaks

This program has been tested for memory leaks using valgrind with the flag `--leak-check=full` and there are no reported memory leaks during the normal execution of this program as well as exiting on error conditions.